CDC® CYBER 200 OPERATING SYSTEM 1.4

FOR USE WITH
CDC® CYBER 200
OPERATING SYSTEM

Volume 1 of 2

REFERENCE MANUAL

CONTROL DATA

60457000

CDC® CYBER 200 Operating System 1.4

Manual Title Reference Manual Vol. 1 (of 2) Pub. No. 60457000 A Rev. A

As part of Control Data's continuing quality improvement program, we invite you to complete this questionnaire so that you may have a more direct influence on the manuals you use.

Please rate this manual for each general and individual category on a scale of 1 through 5 as follows:

1 - Excellent 2 - Good 3 - Fair 4 - Poor 5 - Unacceptable

I. Writing Quality

- A. Technical accuracy _____
- B. Completeness _____
- C. Audience defined properly _____
- D. Readability _____
- E. Understandability _____
- F. Organization _____

II. Examples

- A. Quantity _____
- B. Placement _____
- C. Applicability _____
- D. Quality _____
- E. Instructiveness _____

III. Format

- A. Type size _____
- B. Page density _____
- C. Art work _____
- D. Legibility _____
- E. Printing/Reproduction _____

IV. Miscellaneous

- A. Index _____
- B. Glossary _____

V. Please provide a yes or no answer regarding manuals in general:

- A. I prefer that a manual on a software product be as comprehensive as possible; physical size is of little importance. _____
- B. I prefer that information on a software product be covered in several small manuals, each covering a certain aspect of the product. Smaller manuals with limited subject matter are easier to work with. _____
- C. I am interested primarily in reference manuals designed for ease of locating specific information. _____
- D. I am interested primarily in manuals designed to teach the user about a product or certain capabilities of a product. _____

VI. We recognize that we have a wide variety of users. Please identify your primary area of interest or activity:

- A. Student _____
- B. Applications programmer _____
- C. Systems programmer _____
- D. How many years programming experience do you have? _____
- E. What languages
  - 1. Algol _____
  - 2. Basic _____
  - 3. Cobol _____
  - 4. Compass _____
  - 5. Fortran _____
  - 6. PL/I _____
  - 7. Other _____
- F. Have you ever worked on non-CDC equipment? _____
  - 1. If yes, approximately what percent of your experience is on non-CDC equipment? _____
  - 2. How do you rate other vendor manuals against this and all other CDC manuals using the 1-5 ratings. (Example: XYZ Corp. 2 means XYZ manuals are good as compared to CDC manuals.)

	This Manual	All CDC Manuals
Burroughs	_____	_____
DEC	_____	_____
Hewlett-Packard	_____	_____
Honeywell	_____	_____
IBM	_____	_____
NCR	_____	_____
Univac	_____	_____
Other _____		

General Comments _____

_____

OPTIONAL: _____ _____

Name Company/School

_____

Address

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
4201 Lexington Avenue North
Arden Hills, Minnesota 55112

CUT ALONG LINE

FOLD FOLD

TAPE TAPE

CDC® CYBER 200 OPERATING SYSTEM 1.4

FOR USE WITH
CDC® CYBER 200
OPERATING SYSTEM

Volume 1 of 2

REFERENCE MANUAL

CONTROL DATA

60457000

REVISION RECORD	
REVISION	**DESCRIPTION**
A	Original release.
(7-2-79)	
Publication No. 60457000	

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
4201 Lexington Avenue North
Arden Hills, Minnesota 55112

or use Comment Sheet in the back of this manual

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

Page	Revision
Cover	–
Title Page	–
ii thru xi	A
1-1 thru 1-5	A
2-1 thru 2-9	A
3-1 thru 3-11	A
4-1 thru 4-29	A
5-1 thru 5-13	A
6-1 thru 6-24	A
7-1 thru 7-15	A
8-1 thru 8-10	A
A-1 thru A-4	A
B-1 thru B-28	A
C-1 thru C-3	A
D-1	A
D-2	A
Index-1 thru -3	A
Comment Sheet	A
Mailer	–
Cover	–

Page	Revision

Page	Revision

PREFACE

This manual describes the CONTROL DATA® CYBER 200 Operating System for the CONTROL DATA® CYBER 200 Computer System. The CYBER 200 configuration supports either a CONTROL DATA® CYBER 200 Link Station running under control of the NOS/BE 1 or SCOPE 3.4 Operating System, or a CONTROL DATA® CYBER 200 Access Station running under control of the NOS 1 Operating System.

This manual is published in two volumes:

- Volume 1 is written for applications programmers who will be accessing CYBER 200 Operating System capabilities through batch jobs or interactive terminals. A detailed knowledge of basic programming principles is assumed. Subroutines that interface with the Operating System through CONTROL DATA® CYBER 200 FORTRAN calls or CDC® CYBER 200 Assembler language calls are included. File concepts and debugging information are also presented.
- Volume 2 is written for systems programmers who will be modifying or expanding the capabilities of the CYBER 200 Operating System, and who wish to do so by writing programs that communicate directly with other programs and with the virtual system part of the operating system. All virtual system messages available for writing such programs, as well as some of the system tables referred to by these messages, are described. It is assumed that the reader has some understanding of the principles of operating systems in general.

Related information can be found in the following publications.

Publication	Publication Number
CDC® CYBER 200 Operating System Reference Manual Volume 2	60457010
CDC® CYBER 203 Computer System Hardware Reference Manual	60256010
CDC® CYBER 200 Operating System Operator's Guide	60457030
CDC® CYBER 200 Operating System Installation Handbook	60457020
CDC® CYBER 200 Access Station Reference Manual	60452800
CDC® CYBER 200 Link Reference Manual	60457060
CDC® CYBER 200 Link Operator's Guide	60457070
CDC CYBER 200 FORTRAN Language Reference Manual	60457040
CDC® CYBER 200 Assembler Reference Manual	60457050

CDC manuals can be ordered from Control Data Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTENTS

APPENDIXES

INDEX

FIGURES

TABLES

NOTATIONS USED IN THIS MANUAL

UPPERCASE	Words or character strings that must be entered as shown. They must be spelled correctly including any = or / shown.
UNDERLINED UPPERCASE	Words or character strings that can be abbreviated to the number of underlined characters.
Lowercase words	Generic terms which represent the parameters or character strings supplied by the programmer. When generic terms are repeated in a format, a number or letter might be appended.
[ ] Brackets	Optional portion of a format. All parameters enclosed within the brackets can be omitted at programmer option. The brackets are editorial conventions only; they are not part of the format.
{ } Braces	Portion of a format in which only one of the vertically stacked items can be used. The braces are editorial conventions only; they are not part of the format.
. . . Ellipses	Repetition indicator. The portion of the format immediately preceding can be repeated at programmer option.
Δ	Blank indicator. In a format, this character indicates that a blank or space should appear.
#	Numbers used in this manual are decimal unless noted as hexadecimal. Hexadecimal numbers are prefixed by the # character.

Punctuation characters shown within the formats are required unless the text indicates another punctuation character can be substituted.

INTRODUCTION TO CYBER 200 1

The CDC® CYBER 200 operating system (CYBER 200 OS) consists of a group of programs that comprise the operating system for the CDC® CYBER 200 Computer System, a virtual memory computer. The functions of input, compilation or assembly, loading, execution, and output of all programs submitted to the computer, as well as allocation of main memory, are monitored and controlled by CYBER 200 OS.

SYSTEM CONFIGURATION

The minimum CYBER 200 configuration consists of a central processing unit (CPU) interconnected to the standard magnetic core storage (MCS) unit and four input/output channels.

The computation unit of the CYBER 200 is an autonomous central processor with input/output channel connections. Stations (which consist of a small processor, display/keyboard unit, small drum, and a buffer memory) handle peripheral processing functions and are linked to the CYBER 200 central processor. Slow-speed input/output devices, terminals, magnetic tapes, and so forth, are grouped and connected to stations. A station consists primarily of a small processor designed for data handling, rather than data processing; each station has its own storage system and channels for handling the particular set of devices attached to it. The layout of a CYBER 200 system, with the connections between the various functional units, is shown in figure 1-1.

The central processor contains all streaming and instruction control, arithmetic units, storage access control, and input/output communication control. The standard MCS contains 524 288 64-bit words of storage. The MCS has eight sections; each connects to 132-bit read and write data buses (128 data and 4 parity bits). Each MCS section contains four banks, giving a total of 32 multiphased banks. An optional MCS, identical to the standard, allows expansion to a total system capability of 1 048 576 64-bit words.

The hardware mechanisms of CYBER 200 manage the 64-bit words of main memory in blocks known as large pages and small pages. A large page has 65 536 words while a small page has 512 words. The 524 288 word memory is allocated into eight large pages or 1024 small pages, or is partitioned in a combination of large and small pages.

One of the input/output channels is connected to a Maintenance Control Unit (MCU). The Maintenance Control Unit consists of a station processor having maintenance control and monitoring capabilities.

Each of three additional standard input/output channels provides 16-bit data communication and control to a station processor. Each station processor has a buffer controller and control circuitry connected to the corresponding peripheral equipment. Flexibility in the selection of peripheral equipment connected to the buffer controller is possible because the station operating system software is modular, and only relevant portions are loaded into a particular station processor. (A typical station processor might be connected to a line printer, a card reader, and a card punch.) Eight additional input/output channels (in groups of four) can be added to the system, for a total of 12.

A CYBER 200 CPU, with its immediate storage, can be likened to a data processing station within the system but with no particular priority over any other station. Two other stations, however, are closely associated with the CPU. The Mass Storage Station provides temporary storage for programs exceeding available main memory. The Maintenance Control Unit, in addition to its functions of off-line fault diagnosis/repair and preventive checking, is capable of collecting detailed information about system performance.

The CPU of the CYBER 200 is the computational facility of the system. It is both a conventional scalar processor that operates on two operands at a time and a vector processor that operates on strings of operands. Vectors are contiguous sets of bits, bytes, half words, or full words in virtual memory. Some of the vector instructions act as hardware macros for functions such as polynomial evaluation, byte editing, scalar product of two vectors, and sequencing by merging byte string records.

The portion of the CPU that performs all vector instructions and all operations of operands in floating point format is known as the floating point pipelines. The two floating point pipelines can operate with either 32-bit or 64-bit numbers in floating point format.

STATION DESCRIPTIONS

Stations are the interface between peripheral devices and the main memory associated with the CPU. In general, a station consists of a separate computer system with its own software and operator keyboard/display console. First level stations are connected directly to the main memory. Stations connecting to main memory through another station are second level stations.

The equipment attached to a station is configured to perform specific functions. Station types include:

Mass Storage Station

Service Station with attached Unit Record Station and/or Access Station

Magnetic Tape Station

CYBER 200 Link Station

A system can incorporate a CYBER 200 Link Station and/or an Access Station. Each system configuration has one station, known as a Maintenance Control Unit (MCU), that is used to initiate the operating system (autoload and recovery) and to monitor hardware functioning.

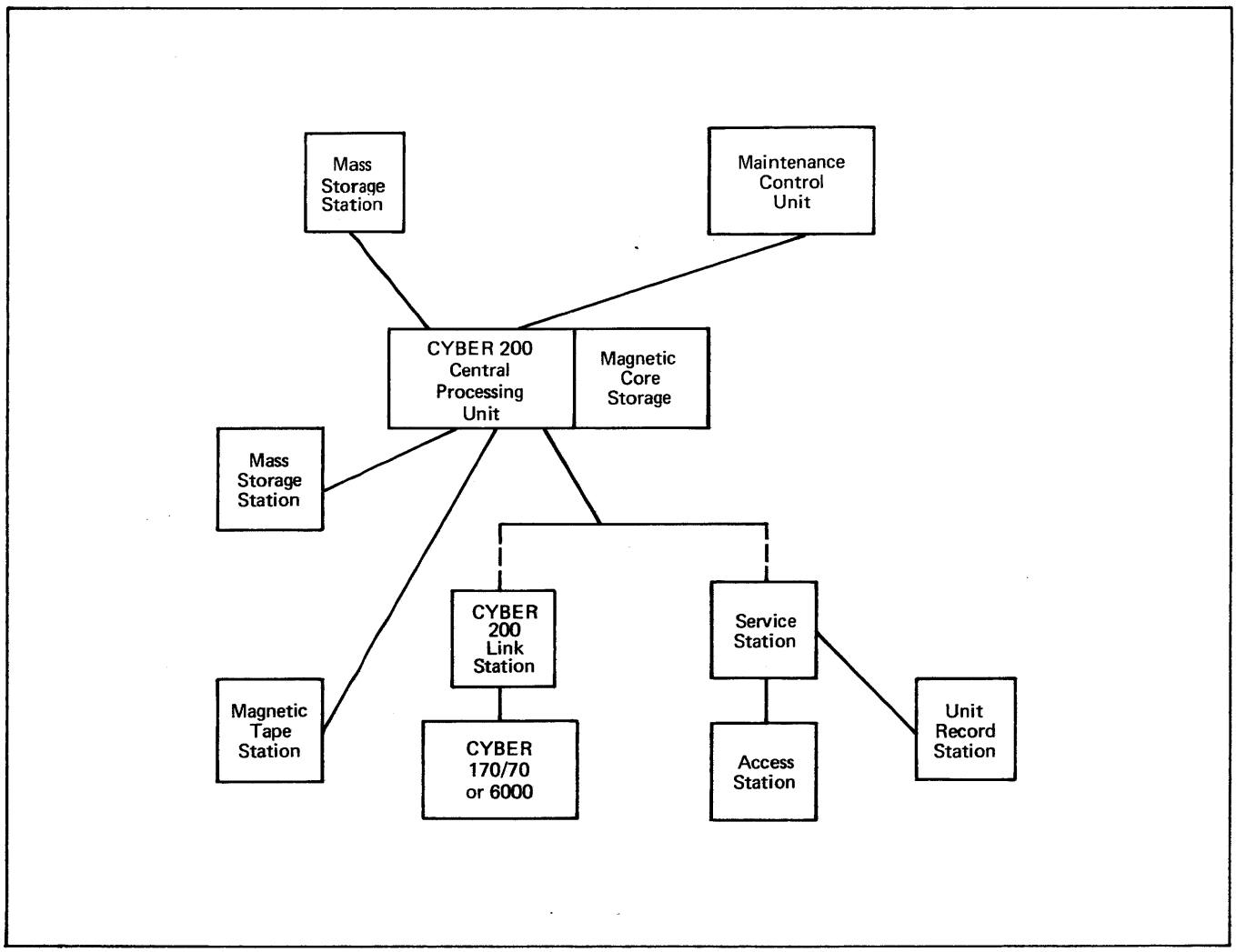


Figure 1-1. CYBER 200 System Showing Component Connections

MASS STORAGE STATION

The Mass Storage Station is the auxiliary memory of the system. The usual storage device is a CONTROL DATA 819 High Capacity Disk (HCD). Up to 8 819 disks might be connected to the station. These devices are used to hold the user's files that are paged to and from the memory. The Mass Storage Station is connected directly to the main memory. The CYBER 200 Operating System can be paged to an 819 HCD.

SERVICE STATION

The Service Station provides a focal point for the Unit Record Station and/or Access Station through which user jobs enter and leave the system. The Service Station controls communication between CYBER 200 and other processors in the input/output network, maintaining message and data paths. At least one 844 Disk Storage Unit is attached to the station to provide temporary storage for files, including the queues of print, punch, and input files. Some files used by the operating system reside on this disk and are paged to and from main memory as needed. Processing by the Service Station includes the validation of users attempting to access the CYBER 200.

UNIT RECORD STATION

The Unit Record Station (URS) is the interface between a 405 Card Reader, 415 Card Punch, and/or a 512 Line Printer and the Service Station. Files transfer between the input/output devices and the 844 disk of the Service Station under control of the Unit Record Station. Processing in the Unit Record Station includes device polling, data compression and expansion, and processing of the card reader identification (STORE) card that begins every card deck.

ACCESS STATION

The Access Station is any CDC CYBER computer system capable of running under control of the NOS 1 operating system. Any of the unit record devices, magnetic tape units, or terminals supported by the operating system can be attached to the Access Station. Files transfer between the Access Station and the disk of the Service Station.

MAGNETIC TAPE STATION

The Magnetic Tape Station maintains files on 657 or 659 Magnetic Tape Units. Processing by the station includes reading or writing tapes and translating character codes.

CYBER 200 LINK STATION

The CYBER 200 Link Station is connected to a CDC CYBER 170 Series, CYBER 70 Models 72, 73, or 74, or 6000 Series Computer System running under control of the NOS/BE 1 or the SCOPE 3.4 operating system. The unit record devices, magnetic tape units, or terminals supported by the NOS/BE operating system can be accessed by the CYBER 200 Link Station.

SYSTEM OPERATOR CONSOLE

The computer operator monitors and controls interactive and batch tasks running under CYBER 200 OS by logging in at a CYBER 200 terminal and executing a file having the task name OPERATOR. This file can be executed only by a user supplying an operator id defined by the installation. Any terminal logged in with the proper id and executing the operator file can be considered as the system operator's console. The terminal remains in this mode until the operator terminates the OPERATOR task or disconnects the terminal. The operator communicates interactively with the OPERATOR program to request information from the system, respond to user requests for equipment, make general announcements, and so forth.

Only one terminal can be the system operator console at any one time. A detailed description of the system operator console is given in the CYBER 200 Operator's Guide.

CYBER 200 OS ARCHITECTURE

The CYBER 200 is a large scale, high-speed computer with a significant number of new developments. Most important are the architectural concepts: virtual memory operation and string array processing capability. Further, the system employs many advanced design concepts, such as stream processing, a large high-speed register file, high input/output channel capacity, virtual bit addressing, large memory bandwidth, and a powerful instruction repertoire.

VIRTUAL MEMORY

Through the virtual memory system of the CYBER 200, an apparently unlimited memory structure can be viewed as if it were entirely main memory. The CYBER 200 hardware mechanisms manage system information in large pages (blocks of 65 536 words) or in small pages (blocks of 512 words). The programmer determines the block size for a given application. The system ensures that the most frequently accessed pages exist in main memory, while unused pages are sent to slower backup media as necessary.

Virtual addresses are contained in a 48-bit format. When 512-word pages are addressed, the virtual page identifier is contained in 33 of the 48 bits; for large pages, the virtual page identifier requires only 26 of the 48 bits. Because unused virtual space imposes no burden on the system, the user can organize program addresses in almost any convenient manner.

Virtual addresses comprise the set assumed to exist by the programmer. Virtual addresses are translated into physical memory addresses by system software as needed when code is brought into the CPU. The system keeps track of the relationship between physical memory addresses and virtual memory addresses through a translator, called a page table. Each entry in the page table contains the virtual page address and the corresponding physical memory address, together with an access mode lock and other control information. A successful association between a virtual address and an entry in the page table causes that entry to be moved to the head of the table; all entries in between are moved down by one place. In CYBER 200, the first 16 entries in the table are kept in high-speed registers; the registers are examined in parallel with a simultaneous associative compare. An unsuccessful compare results in a sequential search through the remainder of the table held in main memory. Addresses of infrequently used pages automatically float to the end of the table.

If an address has no entry in the page table, various hardware sequences are initiated, and the program requesting the address is interrupted. Normally, the system provides the space addressed by requesting the desired page be moved to main memory from a Mass Storage Station. The program continues processing from the point of interruption. The user is unaware of these interruptions.

The virtual memory and the associated paging scheme of the operating system mean that the programmer does not have to break programs into overlays or segments to fit them into main memory. The operating system manages the allocation of storage between main memory and auxiliary memory, moves information from auxiliary to main memory, and translates virtual memory addresses to physical addresses in main memory. However, main memory is where a program executes, and therefore must be taken into consideration when the logical flow of the program is developed.

The operating system considers every program to be executable only in virtual memory. Data files can also be defined by a set of virtual addresses. Each active program in the system executes in its own virtual address space and is protected from other users by a series of hardware and software locks and keys.

STRING ARRAY PROCESSING

The CYBER 200 CPU includes several classes of instructions that can be used for conventional computing or string array processing. Conventional processing is performed by major high performance facilities that operate on floating point operands (64 or 32 bits) and on single bytes and bits. Some floating point instructions operate register-to-register; other operations on words, single bits, or bytes are storage-to-storage.

The CYBER 200 is also a vector processor. In CYBER 200, a vector is defined as a contiguous set of bits, bytes, half words, or full words in virtual memory. The definition depends on how the contents of the virtual space are treated by the vector processing instructions, rather than the nature of the contents.

Pipeline units operate on strings of operands: 64- or 32-bit word arrays, byte strings, and bit strings. Information to specify the addresses of source and destination streams usually is held in the register file. Memory system design accommodates two input operand streams and one output stream simultaneously.

Many user functions provided by the string and array mechanisms perform more complicated operations on streamed data. Such functions amount to hardware macros and include, for example, polynomial evaluation,

byte editing, scalar product of two vectors, sequencing by merging byte string records, and vector arithmetic on sparse vectors.

Each operand for a string instruction can be a vector with as many as 65 536 elements. Although the function is executed serially in a pipeline, it can be considered as being carried out in parallel on the data. Thus, the user has from one to 65 536 parallel processors at his disposal, depending upon how he achieves processing through code selection. Processing facilities allow for efficient computing in the conventional sense, and also provide a fundamentally different approach to programming through string and array processing.

THE REGISTER FILE

The CPU contains a file of 256 64-bit word addressable registers which are used for instruction and operand addressing, indexing, field length counts, and as a source or destination for register-type instructions. The register file is accessible to assembly language programs and to FORTRAN language programs which use special calls. Its contents are, by convention, divided into several areas that can be used to pass parameters to another routine, access data for programs, trace execution, and hold constants. The contents of the register file are dumped to an output file as a standard part of abnormal job termination, and a similar dump can be obtained during program execution through the debugging facilities available in the system.

OPERATING SYSTEM

The CYBER 200 operating system is divided into four parts: the resident system, the virtual system, privileged user tasks, and the peripheral system.

The central operating system and the peripheral operating system each consists of a resident system and a nonresident set of callable tasks. Various portions of the system communicate through messages.

The resident system runs in monitor mode; it is always resident in main memory and references memory by absolute addresses, rather than through the virtual paging mechanism. When the CPU is in monitor mode, interrupts are inhibited, and some extra instructions are enabled.

The virtual system tasks run in job mode and reference memory by virtual address. They communicate with the resident system by using reserved messages, and they can modify system tables. The nonresident set of central tasks comprises the virtual system; it controls the allocation of resources to jobs. In addition, the virtual system contains such functions as file management, explicit input and output, and terminal message processing.

The central operating system requires at least one large page for execution.

Privileged user tasks have the same characteristics as virtual system tasks, but they cannot modify system tables directly.

The peripheral operating system runs in the station processors.

RESIDENT SYSTEM

The resident central operating system has two parts: KERNEL, responsible for time-slicing and message handling, and PAGER, responsible for memory management and page swapping.

All access interrupts, as well as certain messages dealing with memory allocation, are passed to PAGER by KERNEL. PAGER dynamically allocates both large and small pages and performs all required implicit input/output necessary to free memory pages and obtain the pages causing access interrupts.

PAGER determines dynamically which pages of a user's virtual address space have the most activity. These pages define the working set of a program at an instant in time.

VIRTUAL SYSTEM TASKS

The virtual portion of the operating system controls the entry of interactive users and batch jobs into the system, ordering jobs by priority, and entering and removing jobs. In addition, it contains routines for system file management, explicit input/output, and terminal message handling.

PRIVILEGED USER TASKS

A special class of tasks which run under privileged user numbers (privileged user tasks) includes file routing and disposition.

Any task that runs under a privileged user number is a privileged task. Privileged user tasks have basically the same characteristics as virtual system tasks in that they run in job mode, make resident calls, are pageable, access other user files, and reference main memory by virtual address. Unlike virtual system tasks, these tasks do not have direct access to system tables; however, through special calls to the virtual system, they are able to obtain indirect access to system tables. These special system-level calls are available only to privileged tasks.

Because a privileged user can make most resident system calls, such users are able to perform some tasks for the virtual system. This results in a reduction of virtual system overhead and frees the virtual system to process other functions. Tasks such as handling input and output files and operator communication are currently done by privileged user tasks.

PERIPHERAL SYSTEM

The peripheral system for all station processors has two parts:

A resident basic system called NUCLEUS, common to all stations.

A set of overlays for performing tasks for the individual stations.

NUCLEUS is controlled by SCANNER. SCANNER uses scan bits and an associated table to determine which routines to call. If a particular routine is not in core, a resident overlay driver calls in the routine from the station's microdrum.

NUCLEUS uses a set of nonresident tasks to control peripheral equipment. Operating system tasks for each station processor are stored on its own microdrum.

NUCLEUS consists of diagnostic routines, a system peripheral deadstart program, driver programs for the microdrum and keyboard/display, an organizational program, programs to manage the system overlay mechanism, and SCANNER, which is the main control and organizational program.

Nonresident tasks are concentrated into larger processing routines to facilitate on-line error handling and maintenance procedures common to all stations. Further, station functions are grouped into different systems to minimize system tables. Any one system contains only those routines necessary to its operation.

FILE ORIENTATION

CYBER 200 Operating System allows two modes of input/output, explicit and implicit. Explicit input/output provides a conventional manner of data transfer between buffers in main memory and tape or mass storage.

Implicit input/output is accomplished by the operating system when the user causes an access interrupt by referencing a page of data or code not in main memory. If the virtual page has been previously associated with physical space, the system transfers the data between main memory and the physical device. If a virtual-to-physical relationship has not been established previously, the system defines the virtual page in free space so that it becomes an extension of program space. When doing implicit input/output, the virtual address can be considered a symbolic reference to the mass storage auxiliary memory.

CYBER 200 OS recognizes virtual files and physical files. A virtual file is prefaced by a 512-word block containing control information to be used by the operating system. This preface is known as the minus page.

The system allows physical and virtual files to be opened in either the implicit or explicit mode.

Only virtual code files that have been processed by the loader can be submitted to the system for execution. In this case, the loader is responsible for generating the information in the minus page for a virtual code file.

A program can be removed from main memory to facilitate memory management. Corresponding mass storage space is provided for each virtual address space. As a program is put into execution, the operating system automatically creates a file to contain any modified pages of the program file, any free space attached, and any read-only data space defined to have temporary write access. This file is known as the drop file.

SYSTEM USAGE

A programmer can use the CYBER 200 in either a batch mode or an interactive mode. In either case, all that is required is a legitimate user number and account identifier from the operations facility of the desired CYBER 200 site.

An efficiently coded application has its data organized so that it can take advantage of the streaming and vector processing capabilities of the CYBER 200. It also has a well-behaved working set which resides in main memory such that the program can efficiently migrate through the working set.

The user should also give some serious consideration as to whether large pages or small pages are used for the program and data.

CYBER 200 COMPARISON

Significant differences exist between the hardware and software of the CYBER 200 and any of the Access Station or CYBER 200 Link Station systems. Some of these differences that affect applications programs are described below.

1. CYBER 200 memory words are 64 bits.

2. CYBER 200 uses a hexadecimal, rather than an octal, number system. The hexadecimal number sequence is: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

3. CYBER 200 character data is a subset of 8-bit ASCII, not 6-bit display code. See appendix A.

4. Because the CYBER 200 operating system supports virtual memory, programs do not require overlays or segmentation. No overlay or segmentation facilities exist. However, the programmer writing an efficient program is aware of the program flow and develops the application so that the logical flow of the program lends itself to a well-behaved working set.

5. CYBER 200 batch job decks are similar, although not identical, to batch jobs processed by the other CYBER computer systems. In particular, all batch jobs running under a single user number are processed sequentially, not in parallel.

6. A mass storage file can reside in up to four noncontiguous physical areas of mass storage called segments. Each segment is a contiguous area; all areas are on the same disk.

The CYBER 200 system can be accessed interactively through a terminal or can be accessed through a batch deck. These capabilities are described separately below.

UNIT RECORD STATION CARD READER OPERATIONS

NOTE

The following information describes operations available through a Unit Record Station. See other documentation for information about batch deck structure, permanent file handling, and output files for jobs submitted through a CYBER 200 Link Station or an Access Station.

For accounting information applicable to a particular installation, see a site analyst.

A card reader must be used to initiate all batch operations. The deck submitted can contain either:

- One or more jobs to be executed under control of the batch processor.
- Data to be stored as an unattached permanent file.

These two types of operations are described separately below.

A batch deck must begin with a STORE card. The last card in the deck must have the digits 6, 7, 8, and 9 multipunched in column 1.

The STORE card, which is sometimes termed a card reader identification card, has a column-dependent format as shown in figure 2-1.

The name specified in columns 21 through 28 is assigned to the mass storage representation of the card deck. The file is known, in general, as the batch input file, although portions of it might be known by different names during processing as described below. It becomes an unattached permanent file under the user number specified in columns 7 through 12. Column 49 controls whether an existing permanent file with the same name is destroyed or whether the file is not created when the name in columns 7 through 12 duplicates the name of a permanent file already existing for this user number.

Any file created has a maximum size established by an installation parameter. The file is stored at a security level of 0.

Once the file exists on mass storage, the content of column 34 of the STORE card determines any additional processing:

If column 34 contains a blank, no further processing occurs and the file remains in the system as an unattached permanent file under the specified user number.

If column 34 contains a character B, P, or S, the next card in the deck is assumed to be a job statement and the job is executed under control of the batch processor at security level 2.

The mass storage representation of a deck read into the system depends on the processing selected by the STORE card parameters. The type of data in the cards does not affect file structure. Both binary and ASCII data can be stored in either structured or unstructured format.

If a file is to be processed by the batch processor, it is stored in SRM-structured format as ASCII data. Only the unit separators (#1F) appear within the data as ASCII control characters. All other file divisions are identified by information in the control words that accompany data in SRM-structured files.

If a file is not to be processed by the batch processor, it is stored in unstructured format. Division terminators appear on the file as 8-bit ASCII control characters:

Division	ASCII Control Character
Unit	#1F
Record	#1E
Group	#1D
File	#1C

For the most part, an applications programmer need not be concerned with internal file structure.

CARD FORMATS

A batch deck can contain three types of cards. They are ASCII coded cards, CYBER 200 binary cards, and 80-column binary cards. More than one type of card can appear in a single deck as long as they are separated by separator cards.

ASCII Coded Cards

ASCII coded cards have the punch codes indicated in appendix A. Their most customary content is a source program to be compiled or assembled, data to be processed, and the control statements and directives of a batch job.

ASCII codes can correspond to either the codes of an 026 keypunch or the codes of an 029 keypunch, depending on the installation configuration. Columns 79 and 80 of a STORE card or separator card indicate whether the following ASCII coded cards are 026 or 029 format.

Column	Contents	Description
1-5	STORE	
6	blank	
7-12	6 digits	User number with leading zeros if needed to fill field.
13-20	1-8 characters	1 through 8 character account identifier; can appear anywhere in field.
21-28	1-8 characters	Name (1 through 8 letters or digits beginning with a letter) to be assigned to deck. Should not be the same as any of the following: Public file name. Permanent file of the same user number. Name on any other STORE card that might be in the system at the same time under the same user number.
29	blank	
30	A, S, R, blank	File type. Applicable only when the batch processor is not to process the file. Structure of the file to be created from the card deck: A 80-column binary. S System Record Manager structure. Default if column 34 is not blank. R Record structure. Default if column 34 is blank.
31-33	blank	
34	P, B, S, blank	Job priority indicator: P Priority. S Standby. B Batch. blank Deck is not a job to be executed by the batch processor.
35-48	blank	Unused.
49	C, U, blank	File creation mode: C Conditional. Create only if a permanent file with the same name does not exist for this user number. U Unconditional. Create even if an existing permanent file with the same name must first be purged. Default is an installation parameter.
50-78	blank	
79-80	26, 29	Keypunch indicator: 26 Deck punched on 026 keypunch. 29 Deck punched on 029 keypunch. Default is an installation parameter.

Figure 2-1. STORE Card Format

CYBER 200 Binary Cards

CYBER 200 binary cards each have the digits 7 and 9 multipunched in column 1. They normally contain object code produced by the CYBER 200 FORTRAN compiler or assembler.

Only 76 columns of a card contain data; the remaining columns contain system information such as a checksum and a sequence number. The format of CYBER 200 binary card is shown in figure 2-2. Fields have these meanings:

Byte Count	Number of 8-bit bytes on this card, starting with bytes in column 5. Only the number of data bytes specified by this count is actually written to a disk file as data.
Sequence Number	Sequence number of this card.
Checksum	24-bit arithmetic sum of 8-bit data bytes on this card.
Byte 1, Byte 2, . . .	8-bit data bytes to a total of 912 bits in 76 columns.

80-Column Binary Cards

80-column binary cards are treated as a string of 960 bits of data corresponding to fifteen 64-bit words. All columns in a binary card are copied directly to a mass storage file with no conversion of any kind. A card in this format is also known as an absolute binary card. Bits are ordered as shown in figure 2-2.

Cards in this format must follow a card with the ASCII characters UNFORMAT in columns 1 through 8. A similar card must follow all 80-column binary cards. These UNFORMAT cards must appear in a deck in addition to any separator cards required in the deck.

Separator Cards

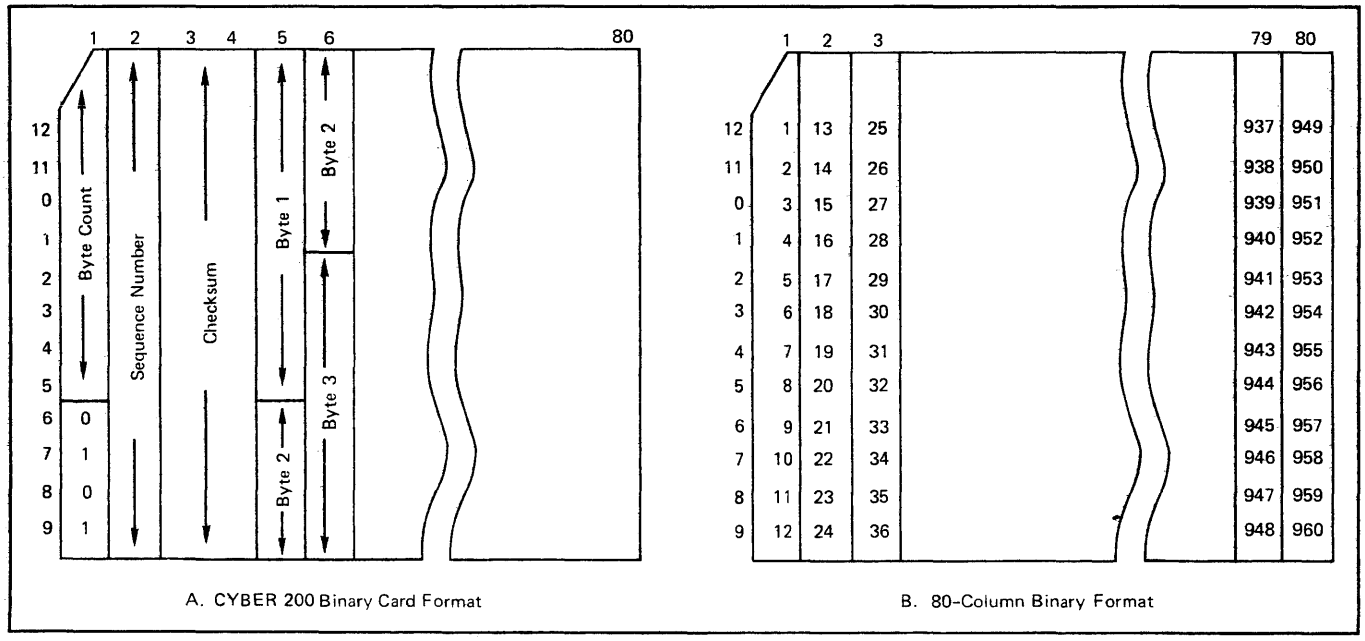
Separator cards are required to separate different parts of a batch deck. During execution of a batch job, the batch processor treats all cards between two separator cards as an individual file. A separator card is required between control statements and a program to be compiled, for instance, and another separator card is required between a source program and any data to be processed during execution of that program.

Separator cards have digits multipunched in column 1. The three separator cards and their uses are:

7/8/9	Record separator that indicates the end of the control statements, source program, directives, or data cards. A name of 1 through 8 ASCII characters can begin at or after column 2 to specify a file name to be assigned to the cards that follow. The file is created as a local file.
6/7/9	Group separator that indicates the end of a job. The card immediately following must be a job statement or a 6/7/8/9 card.
6/7/8/9	File separator that indicates the end of the batch deck.

Keypunch conversion mode can be specified in columns 79 and 80 of a separator card. If specified, the conversion mode remains in effect for all succeeding cards until explicitly modified on a following separator card. The conversion modes are the same (26 or 29) as specified on the STORE card in columns 79 and 80.

Figure 2-3 shows a typical batch deck containing two separate jobs. The first job presumes control statements that compile the source program. The second job presumes control statements that manipulate two separate sets of data. The 7/8/9 card that appears before the 6/7/9 card is optional. Similarly, any 6/7/8/9 card can be preceded by a 6/7/9 card.



A. CYBER 200 Binary Card Format

B. 80-Column Binary Format

Figure 2-2. Binary Punch Card Formats

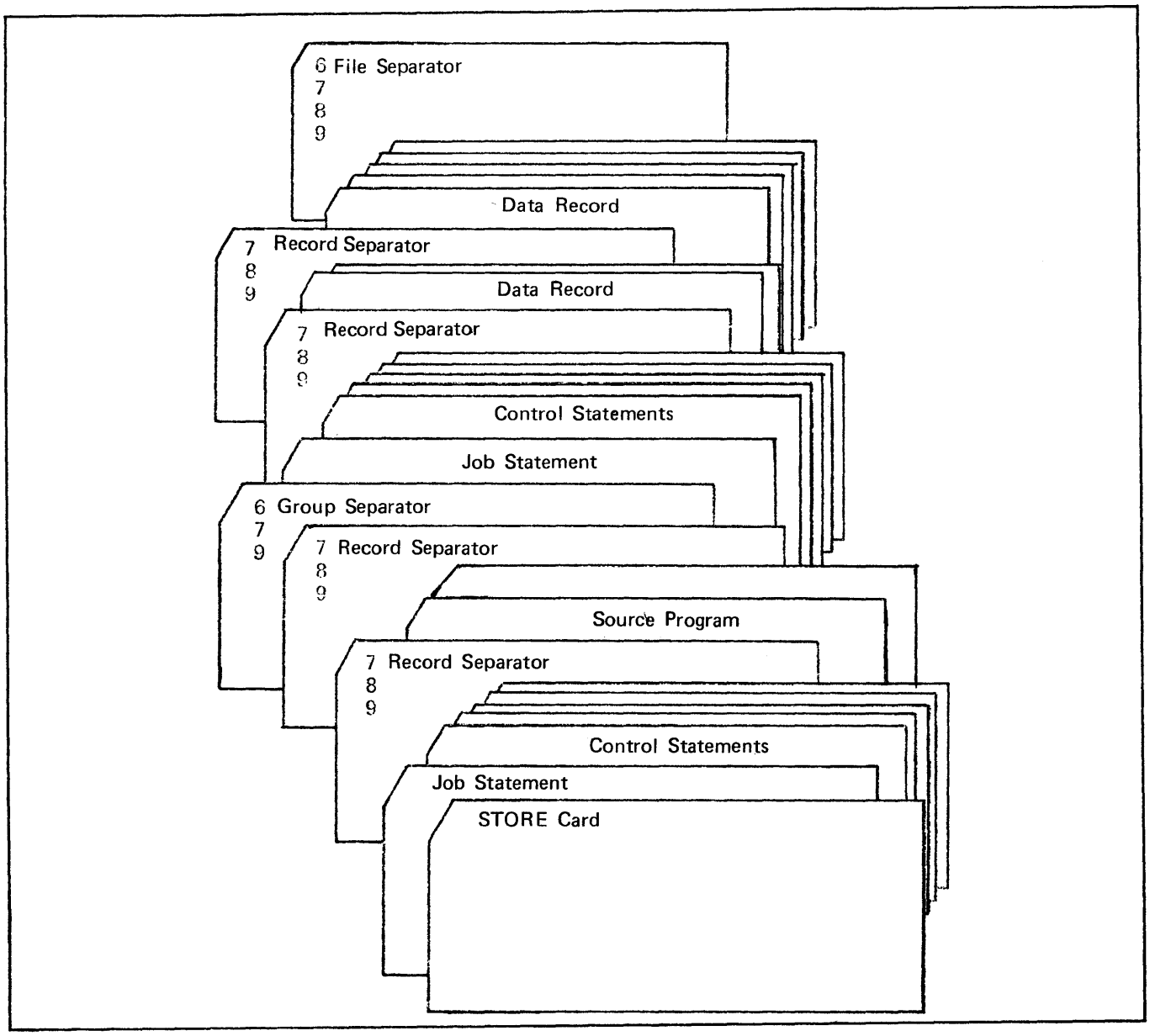


Figure 2-3. Example of Batch Deck with Two Jobs

PRIVATE FILE CREATION

When the STORE card of a batch deck contains a blank in column 34, the batch processor is not called into execution. Rather, the cards in the deck are stored as an unattached permanent file in unstructured file format.

The file has ownership category private, which means it can be accessed by a batch job or interactive user only under the user number by which the file was created.

Any file created by a user task must have a name of 1 through 8 letters or digits beginning with a letter. System-created file names can begin with any legal character.

JOB STRUCTURE

A job begins with a job statement; a job ends with a 6/7/9 separator card or with a 6/7/8/9 separator card, whichever comes first. A job must exist within the context of a batch deck beginning with a STORE card.

A job contains a control statement record that must immediately follow the job statement. Source programs to be compiled or assembled, data to be processed, or directives to be used during control statement processing can also appear in a job. Each of these elements must be preceded by a 7/8/9 separator card.

The control statement record contains one or more control statements. Within a job, control statements can be one of two types:

- Control statements that specify the name of a task to be executed as a controllee of the batch processor. The task might be one of the utilities available as a public file to all jobs; it might alternatively be the name of a controllee file produced by a control statement that called the system loader to create that file. These statements correspond to those that could also be executed through an interactive terminal.
- Special control statements that specify an action to be performed by the batch processor directly. They are valid only in a batch job.

Both types of control statements are described in section 4.

The six special control statements that give a programmer control of the environment of a batch deck and a batch job are:

TV	Redefine error return code threshhold for job termination.
EXIT	Establish control path for abnormal job termination procedures.
READCC	Read control statements from another file.
COMMENT	Send message to job dayfile.
RERUN and NORERUN	Set or reset rerun indicator for entire batch deck to determine whether the file indicated by STORE is to be destroyed or rerun in the event of system failure.

The batch processor attempts to process control statements in the order that they appear in the control statement record. If an executable file (virtual code file) is not found after searching available files, the batch processor aborts the job. Virtual code files are searched in the order: local, attached permanent, attached pool, and public. Data files are ignored when searching for an executable file.

A named record exists when a 7/8/9 separator card has a name punched in it. The batch processor treats the cards following the 7/8/9 separator card as a separate file with the name specified, returning any existing file with that name, if necessary, to create the new local file from the cards. Such a named record does not become a file with the name INPUT as described below. Rather it retains its specified name and can be referenced by tasks in the job.

JOB PROCESSING

The following information describes the way the batch processor executes. The batch deck of figure 2-4 is presumed.

The batch input file, as such, does not have the name INPUT. Several files by the name INPUT might be created (REQUESTed) and destroyed (RETURNed) during execution. Similarly, several files with the name OUTPUT can be created (REQUESTed) during the job's execution.

```
STORE nnnnnnaaaaaaaaMYJOB002        B
MINE02,TV0.
FORTRAN.
LOAD.
GO.
DEFINE,DATAOUT.
7/8/9 separator card

    PROGRAM . . .INPUT . . .OUTPUT . . .DATAOUT . . .
    :
    PRINT
    :
    END

7/8/9 separator card

    data cards

6/7/8/9 separator card
```

Figure 2-4. Example of Batch Deck with One Job

When the batch deck enters the system, all cards between the STORE card and the 6/7/8/9 card are stored as an unattached permanent file with the name indicated on the STORE card. The batch processor creates a local file with the name of INPUT from the first unnamed record of the file following the control statements. It also creates a job dayfile with the name Q5DAYFLE. The dayfile is under control of the batch processor alone and consists of the times and events of the job, along with any status, comment, or error information produced during job execution.

The batch processor also creates a local file with the name Q5JOBFLE for its own use.

The batch processor then examines the first control statement. Since the FORTRAN compiler call of figure 2-4 is not one of the special control statements that the batch processor executes directly, it is presumed to be the name of a file in executable format. The batch processor instructs the operating system to execute the file, and any parameters on the control statement are passed to the task. The operating system searches for a virtual code file with the name on the control statement, treating each file ownership category separately and searching in this order:

Attached private files for this user, including local files and attached permanent files

Pool files in the order any pools have been attached by the user

Public files

Once a file with the correct name is found, execution is initiated with the task running as a controllee of the batch processor. In this instance, assuming the user does not have an attached file with the name FORTRAN, the system finds the FORTRAN compiler available as a public file.

By default, the FORTRAN compiler assumes that the source program to be compiled resides on an attached file named INPUT. The compiler uses the file created by the batch processor and creates the local file BINARY as a result of compilation. The compilation process also creates a local file with the default name of OUTPUT to contain the FORTRAN source listing.

When the FORTRAN compiler completes execution, the batch processor examines the termination value returned from that task, comparing it against the TV value of 0 established by the job statement. If the task returns a value greater than 0, the batch processor initiates job termination procedures. Those procedures would entail searching for an EXIT control statement. If the compilation indicated errors (return code 4, for instance), the job terminates.

If the task returns a TV value of 0, the batch processor then examines the file INPUT. Since the file has been used by the task just concluded, the batch processor returns the existing file INPUT. The batch processor then creates a new local file named INPUT from the next unnamed record that follows the control statements.

Notice that as a result of the creation, destruction, and re-creation of the file INPUT, it is not possible for more than one task to reference the same unnamed record in the batch deck.

The file OUTPUT has its name changed by the batch processor to a name that indicates a member of a family of print files. OUTPUT becomes P00hmmss, where hmmss is the time the batch processor is started.

The batch processor then examines the next control statement and either takes the action directed or starts a file with the name on the statement, as before. In this example, the file LOAD is found as a public file. LOAD executes as a controllee of the batch processor.

During execution, LOAD uses a file with the default name BINARY as its input. It creates a local file of executable code that has the default file name GO. It also creates a file with the default name OUTPUT to contain the load map. At the end of LOAD execution, the batch processor again checks the code returned by the task against the termination value of 0, examines the file INPUT, and searches for a file with the name OUTPUT. Since the file INPUT has not been used during LOAD execution, it remains as it was created. The name of the file OUTPUT is changed to the name P01hmmss. The batch processor then examines the next control statement.

In searching for a file with the name on the control statement GO, the system finds the local file GO created by LOAD. GO then executes as a controllee.

During execution of the FORTRAN program, the program is assumed for this example to read from file INPUT and write to the file OUTPUT. Assume it also writes to a file the PROGRAM statement equated with the local file name DATAOUT.

At the end of execution of the task GO, the batch processor finds the file INPUT has been used and returns it. No more records exist to be used to create a file INPUT. The name of OUTPUT is changed to P02hmmss.

The next control statement is DEFINE, which makes the local file DATAOUT a permanent file. The file DATAOUT therefore remains in the system; all other files for the job are destroyed at job termination.

The batch processor then ends the job. As part of job ending procedures, the batch processor changes the name of the dayfile from Q5DAYFLE to PXXhmmss. Adding a file with a name beginning with PXX completes the family, and the family is routed to a line printer. Family files are printed as one file.

The file MYJOB002 is then destroyed.

ACCOUNTING

The operating system provides a set of features that an installation-supplied accounting routine or the operator can use to control system resources consumed by an individual user or a group of users.

The accounting system can be used as follows:

1. A user number is associated with a division and a repository at the time it is authorized system access. In order to access the system, the user must then withdraw an allocation of system resources from the repository before he can use the system under his user number. These functions are performed by the installation-provided software. The system billing unit algorithm is installation defined. The default unit is time in microseconds.

2. Having gained access to the system, a user can then execute tasks and jobs. Statistics are accumulated over each task or batch job in both the cumulative accounting buffer and the accounting file. The installation has the option to use the statistics in the buffer to debit the user's allocation of resources withdrawn from the repository. It can also use the equivalent statistics in the accounting file to charge the user.

INTERACTIVE ACCESS

Interactive access begins with a LOGON command that identifies the terminal user for security and accounting purposes. Access ends with a BYE request line.

Format of the LOGON command is shown in figure 2-5. Blanks separate fields in the line. The NEW-LINE key terminates the entry.

LOGON userno suffix account level

userno	User number of 6 digits that uniquely identifies the terminal user. Leading zeros should be specified if needed for 6 digits.	
suffix	Letter A, B, C, or D under which a subsequent task is to execute.	
account	Account identifier of 1 through 8 characters.	
level	Single character defining a security access level. Optional.	
	Undefined	Level 0
	P or omitted	Level 2
	A	Level 3
	S	Level 5
	K	Level 7

Figure 2-5. LOGON Format

The LOGON command establishes a user number and a suffix for execution of a task. The user number establishes the files that can be accessed. An interactive task can be initiated under any of the four available suffixes.

Four types of entries can be made from an interactive terminal after LOGON:

- A request line that makes a direct request to the operating system. These entries begin with a special character defined by the installation.
- A file name that calls a private file or pool file into execution. The file or pool must be attached and must have controllee file format prepared through the loader.
- A control statement that calls a public file into execution.
- A message to be sent to an executing task.

A programmer can log on to the system under a given user number and suffix, initiate a task, execute a BYE request line, then log on again with a different suffix to continue operations while the first task is executing. A request line can also change a suffix as an alternative to another logon.

If a programmer has both batch job capabilities and interactive capabilities available and in use at the same time, actions of any batch jobs should be considered before interactive tasks are initiated, since permanent file name conflicts might occur. All batch jobs execute under suffix D, which restricts interactive operations to suffixes A, B, and C under the same user number while the batch job is executing.

REQUEST LINES

A request line makes a direct request of the operating system. Each request line is prefaced by a special character that distinguishes it from other types of entries from a terminal. The special character, by default, is the character $, but it might be changed by the installation.

Possible requests are as follows. The special character is indicated by (sc).

(sc)T — Get current date and time in format mm/dd followed by hh.mm.ss.

(sc)S — Get current state of program active under user's suffix; possible responses appear in table 2-1.

(sc)BB — List current accounting information for program active under user's suffix. Remaining time units are displayed.

(sc)? — Get current date, time, accounting information, and program state for program active under user's suffix; equivalent to (sc)T, (sc)BB, and (sc)S.

(sc)SU — List current activity of programs active under all user's suffixes, A, B, C, and D.

TABLE 2-1. PROGRAM STATES

Response	Meaning
RUNNING	In execution.
WAIT ALT	Waiting for CPU assignment.
WAIT TPE	Waiting for tape assignment.
WRT CNTR	Waiting for controller to get on disk.
WRT CNTE	Waiting for controllee to get on disk.
RCV CNTR	Waiting for message from controller.
RCV CNTE	Waiting for message from controllee.
RCV PDP	Reserved for CDC.
SND CNTR	Waiting to send message to controller.
SND CNTE	Waiting to send message to controllee.
SND PDP	Reserved for CDC.
SND OPR	Waiting to send message to the operator.
SND TTY	Waiting to send message to the teletype.
DUMPING	Input/output being dumped to disk.
FINISH	Finished; clean-up is in progress.
SUSPEND	Suspended.
WAIT MP	Waiting for minus page to be assigned.
RCV OPER	Waiting to receive message from the operator.
WAIT mfx	Waiting for mainframe identified.
NIL	No tasks in execution.

(sc)BP — List time remaining in repository to which the user belongs. Time remaining reflects balance after initial time increment is granted at logon, and after any additional time is drawn from the repository.

(sc)G+xx — Draw xx minutes from the repository. Response might indicate no pool to be referenced.

(sc)G-xx — Return xx minutes to the repository.

(sc)U — List time consumed by user since logon.

(sc)PR	List number of job tasks in interactive (I) class waiting for execution.
(sc)I	Send program to current interrupt routine, if program is so enabled.
(sc)OP message	Send message to operator's terminal. When the request has been completed, the system responds with OK .
(sc)suf	Disconnect terminal interface with current suffix and remain active under new suffix suf.
(sc)BYE	End interactive access to CYBER 200.
(sc)P	List attached pools for this user.

The repository that is referenced by several of the request lines is an allocation of time (calculated internally as system time units) that limits the amount of system resources available to an account. An installation can select an accounting system that does not use a repository.

Any tasks active when either the change suffix request line or the BYE request line is entered remain active and continue to the end of execution.

A special BREAK character can be used to terminate the task currently running. The character aborts the current controllee and transfers control to its immediate controller. If the aborted task has no controller, it can be restarted by executing its drop file. The default BREAK character is !, but it might be changed by the installation. When the system starts to break the job, the message BREAK is sent to the task's level-1 controller.

PRIVATE FILE EXECUTION

Any attached private file in controllee file format can be called into execution through an entry that has the general format shown in figure 2-6.

Tasks can be programmed such that they expect messages from the terminal. Messages can have any format. No buffer exists for terminal entries that would allow several messages to be entered before the first is accepted by a task. A second message for an executing task should not be entered until the first is accepted; otherwise, the second might overwrite the first. Programming the task to prompt for input and to acknowledge output can regulate message flow. Messages can be sent to a task only when the task is executing under the suffix currently in use.

Any parameters should conform to the conventions used on system-supplied control statements. All addresses are assumed to be hexadecimal values; any other number is assumed to be a decimal value unless it is preceded by #.

CONTROL STATEMENT EXECUTION

Most of the control statements described in section 4 can be entered through the terminal. The format for interactive and batch use is the same, except where differences are specifically noted.

task-name / t,c / string

task-name	Name of task to be placed into execution. Must be 1 through 8 letters or digits. The first 6 characters must not duplicate those of any other file to be called into execution.
/ t,c /	Optional time and job class information. The character / must be preceded and followed by a space. A space can appear in place of the comma after the time parameter.
t	Decimal number specifying the maximum amount of time a task can run. Time is an installation defined System Time Unit (STU). Default is 1 STU; maximum is 9999 STUs.
c	Characteristic of the task which influences the way in which the system allocates resources to the task:
P	Priority task requiring rapid response. The installation controls whether a user number can issue priority tasks.
I	Interactive task. Default.
B	Task involving little or no communication between the terminal operator and the executing program.
S	Standby task that executes after all tasks of P, I, or B characteristic.
string	Character string to be passed to the task. The format of the character string is dependent on the coding of the task. All characters after the second /, beginning with the first nonblank character, are passed to the task.

Figure 2-6. General Format of Interactive Task Call

Many of the utilities can be called by a complete control statement or by utility name alone. When only the utility name is entered, the utility responds by sending prompting messages to the terminal. In response to the prompting message, the terminal user should enter an option, terminating each entry by pressing the NEW-LINE key. Additional prompting messages can appear.

Figure 2-7 illustrates interactive call of the loader. (lf) indicates the NEW-LINE key. Interactive use of the loader requires special care. The terminal user must type space line feed to indicate no options or when terminating options. The terminal user begins by entering:

```
LOAD (lf)
```

System response is shown in lowercase letters; uppercase letters indicate the terminal user reply to load files XA, XB, and XC and have the loader write the executable virtual code file to TONY with a load map on file PRINTMAP.

input ?	Request from loader
XA, XB, XC (lf)	User enters names of files containing modules output by a compiler
origin ?	Request from loader
28000 (lf)	User enters bit address where first module is to be loaded
entry ?	Request from loader
(lf)	User indicates no option
any other options ?	Request from loader
CN=TONY (lf)	User indicates controllee option to create virtual code file TONY
continue	Answer from loader
OU=PRINTMAP (lf)	User indicates load map file name
continue	Answer from loader
(lf)	User terminates options and calls for start of load operations

Figure 2-7. Example of Interactive LOAD Call

The CYBER 200 operating system can process mass storage files and magnetic tape files.

MASS STORAGE FILES

The operating system is file oriented. All user information in main memory is allocated corresponding storage space on a mass storage device. Tasks can be entered automatically into and removed from main memory by the system as task management requirements dictate.

A file has a single name by which it is known at all times. (A permanent file name versus logical file name distinction does not exist.) The file is accessed by its name.

A mass storage file can reside in up to four noncontiguous physical areas of mass storage called segments. Each segment is a contiguous area; all segments reside on the same disk. Maximum segment size is an installation parameter.

No logical structure is assumed for files passing between mass storage and main memory. All input/output at the operating system level is by physical pages of 512 words. Tasks running in job mode, including the compilers and system utilities, create logical file structures; but the operating system itself imposes no such structure or overhead.

As a consequence of the lack of a required file logical structure, no positioning is possible at the operating system level. The System Record Manager used by the FORTRAN run-time routines defines a file structure that does allow positioning.

Files are divided into three ownership categories: private (including local and permanent), pool, and public. Under each category is a set of extrinsic regulations: owner, user, and access modes, which the disk subsystem uses to ensure the integrity of a disk file.

Access to the file is determined in the first instance by file ownership category. The search hierarchy is shown in figure 3-1. Access is further controlled by the access category (read or write) established by the file owner. No passwords, as such, exist.

FILE OWNERSHIP

File ownership determines who has access to a file. A privileged user has ownership rights over all files except local files belonging to other users. Nonprivileged users have rights that depend on ownership category. Each mass storage file has one of the following ownership categories:

PRIVATE. Private files include local files and permanent files. In the case of nonprivileged users, only the user who created the file, or only the user to whom ownership was transferred, can access a private file. Permanent files are accessible after they are attached.

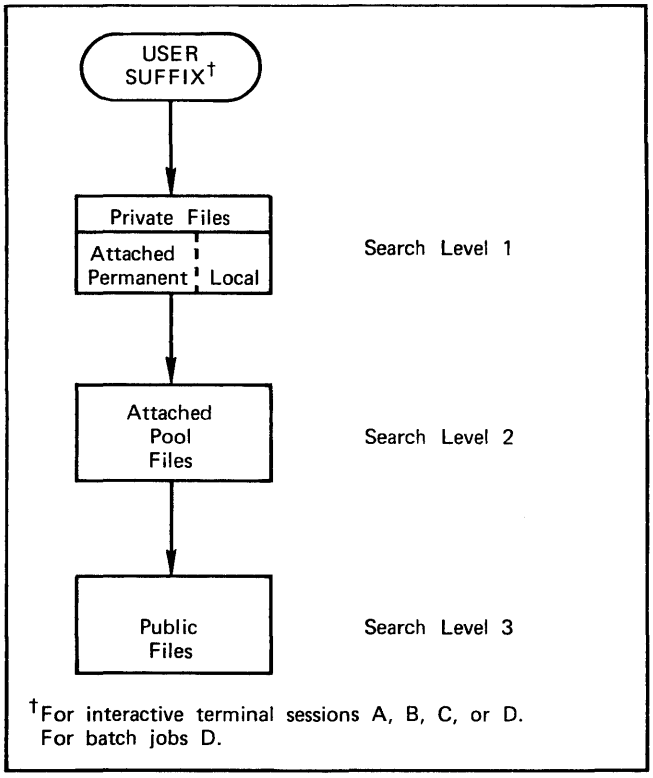


Figure 3-1. File Search Hierarchy

POOL. A group of users can access any file in a pool, as long as the pool boss who created the pool authorized those users by user number. Pool files are accessible after the pool is attached.

PUBLIC. All users can access public files.

File names of files accessible to a user/suffix combination must be unique within each ownership category, but names need not be unique when all categories are considered. A user can create a private file with the same name as a public file, for instance. Two users can each have a private file with the same name, since the category of private exists for each user number. Names of private files accessible to the user must be unique, but as many as five private files with the same name might exist (one unattached permanent file and four local files, each under a different user/suffix combination).

The owner of a file has the right to establish the specifications for a file which, in turn, describe the file. These attributes are described in the DEFINE and REQUEST control statements. The specifications can be temporary for the execution of a given task or can be permanent specifications. The user of a file can be a single user, all users, or a group of users.

Some characteristics of the ownership categories are listed in table 3-1. File ownership categories are illustrated in figure 3-2. The control statements are described in section 4.

TABLE 3-1. FILE CHARACTERISTICS BY OWNERSHIP CATEGORY

Characteristic	Ownership Category		
	Private (including local and permanent)	Pool	Public
Dynamically created	Yes	Yes	No
Accessible globally	No	†	Yes
Multi-user accessible	No	†	Yes
Dynamically updatable	Yes	Yes	No
Dynamically purgable	Yes	Yes	No

†Yes, if pool boss authorizes.

Private Files

A private file belongs to a specific user. Private files include permanent files and local files. The term permanent file implies ownership category private. Permanent files exist until they are explicitly destroyed. Pool and public files must also be explicitly destroyed but are not called permanent files.

A user's permanent files are brought into a system table of active files known as FILEI as soon as the user becomes active in the system by logging on an interactive terminal or running a batch job. Permanent files must then be attached in order to be accessed. To avoid conflicts between jobs, only one suffix can attach a particular permanent file at one time.

Local files exist only at the suffix level and are at the same search hierarchy level as attached permanent files. Local files are created and used only for the duration of the terminal session or the batch job. Local files created with the same name but under different suffixes do not conflict, and names of local files do not conflict with the names of unattached permanent files. Unless explicitly made permanent by the user, local files are destroyed at the end of a batch job or interactive terminal session.

The term attached private files refers to local files and attached permanent files.

Private files are accessible only to the current owner and (for permanent files) to privileged tasks. Only the current owner has control over the files. The owner can access the file, manipulate the contents, change the file access, security level, and retention period of the file, and so forth. A nonprivileged user can give any attached private files to any other user, except to user number 000000.

Each private file cataloged in the file index table belongs to a particular user number and account identifier. When a private file is given by one user to another, the user number associated with the file is changed immediately. The account identifier is not changed until the file is referenced by the receiver. The system accounting tables then indicate the total time that file ownership was held by the originating account identifier.

Pool Files

A pool file is a file with ownership category pool. A pool is a set of pool files. A pool is available for use after the following steps are performed:

The pool utility PCREATE is called to define a pool name.

The GIVE utility with the P=poolname parameter is called to enter one or more existing files into the pool.

The user defining the pool name is known as the pool boss. A pool boss has control over the pool. Authorized users can access files in the pool, but only the pool boss can delete files from the pool or destroy the pool. Use of the pool utility PACCESS determines whether the pool boss alone, all users, or only particular user numbers can access the pool.

Once a pool exists, any authorized user can access a pool file as follows:

Attach to the pool using the pool utility PATTACH.

Access any pool file within limits established by pool boss. The pool boss can allow the file to be read or written depending on the SHARE parameter of GIVE when the file was made part of the pool.

When the pool is no longer needed, detach the pool using the pool utility PDETACH.

One user can be attached to as many as four pools at the same time. An unauthorized user receives a return code of 8 upon attempted pool access.

Pools are manipulated through seven pool file utilities. Four of these utilities can be called only by the pool boss. They are:

PCREATE	Establish pool name.
PACCESS	Define authorized users.
PDELETE	Remove user authorization.
PDESTROY	Destroy pool.

Two pool utilities can be called by any authorized user:

PATTACH	Attach to pool.
PDETACH	Detach from pool.

The remaining pool utility can be called by anyone:

PFILES	List pools, authorized users, and number of users currently attached to pool.

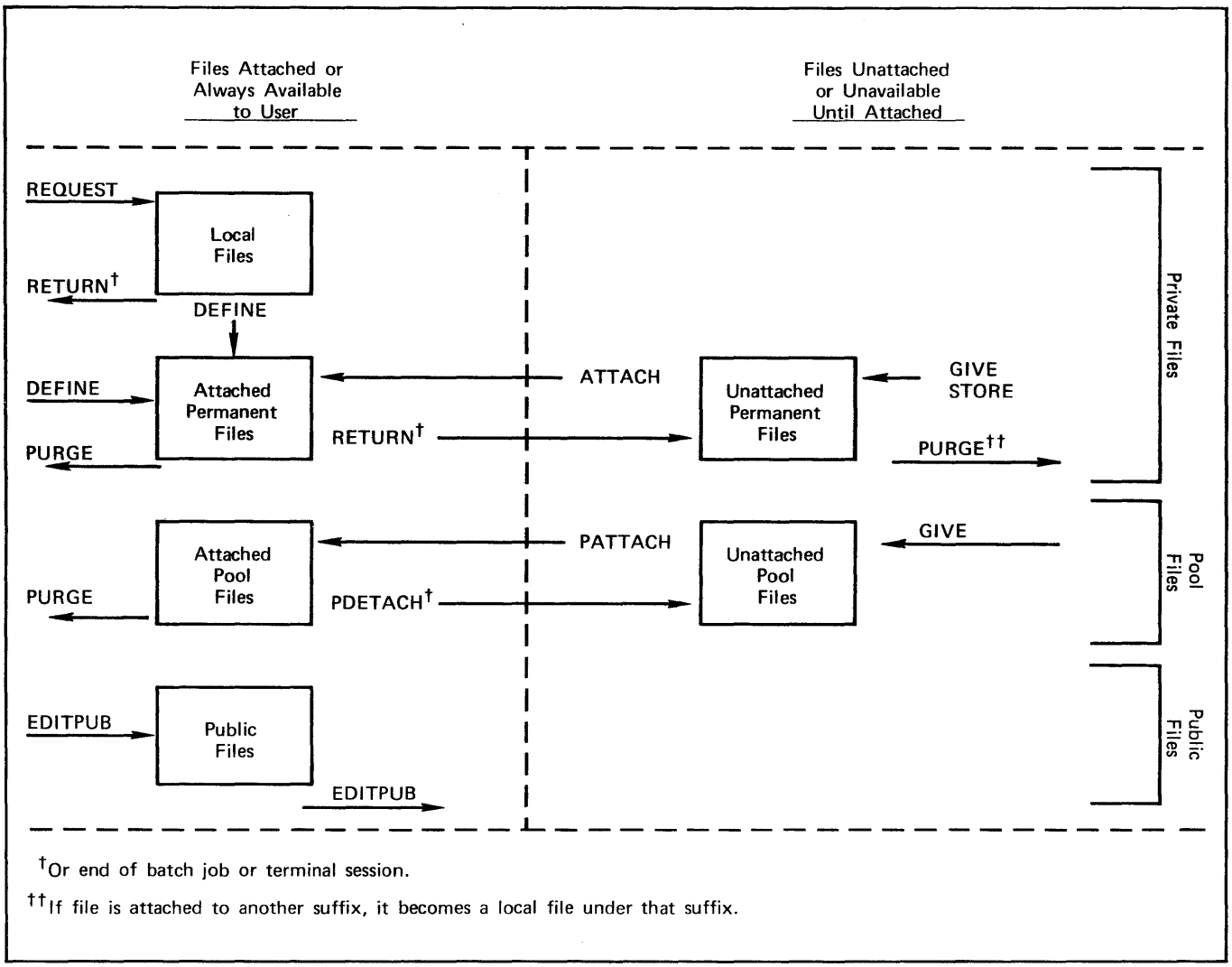


Figure 3-2. File Ownership

Existing pools are maintained by the pool boss using pool utilities to control authorization of users, but using standard utilities to control files in the pool:

GIVE	Add a private file to the pool and specify whether the file can be read or written.
PURGE	Destroy a pool file.

The SWITCH, ROUTE, and EDITPUB utilities cannot be used with a pool file.

Pool is an ownership category just as private and public are ownership categories. File names in the pool must be unique within that pool, but the same file name can be used in more than one pool.

Pool files are associated with a user number and all of its suffixes. When a user number is in use interactively and in batch operations at the same time, batch end-of-job procedures detach the pool from the user number as long as the batch job was the first to attach the pool, whether or not that pool is in use interactively. Pools first attached by an interactive user are detached when the user is no longer active under any suffix.

Public Files

Public files are accessible by the entire body of users. They contain assemblers, compilers, and other general purpose routines that augment the operating system for a particular installation. All utilities described in this manual, for example, exist as public files.

Public file names constitute a kind of job control language for the operating system. All public files have a user number of 000000, signifying ownership by the system. The security level of all public files ranges from 0 through 7.

The list of public files is controlled by the installation administrator or privileged users.

DEVICE TYPE OUTPUT FILES

Output files must be created by the user and given to the system by ROUTE. Output files are further categorized by output device. Each device type output file is processed by a particular privileged task. Table 3-2 shows output file characteristics.

TABLE 3-2. CYBER 200 DEFAULT FILE CHARACTERISTICS

Device	Disposition Code	Internal Characteristics	External Characteristics
Line Printer	PR	PA - Unstructured file with ASCII data and ASCII carriage control characters. Default.	None
		AS - Unstructured file with ASCII data and ANSI carriage control characters.	None
Card Punch	PU	AS - Unstructured file with ASCII data. Default.	29 - 029 Keypunch format (default) 26 - 026 Keypunch format
		BI - Unstructured file with binary data.	80 - 80 column Binary format (default) *B - CYBER 200 Binary format

Files punched by the Unit Record Station (URS) are preceded by two banner cards. The first card contains the user number; the second card contains the file name as described above. Files punched in ASCII are terminated with a 6/7/8/9 separator card.

For the line printer, output files can be saved in families for consecutive processing. A family is Pddfamnm, where dd is either decimal numbers 00 to 99 or the characters XX, and famnm is family name. Family files are held in an unprocessed state until a file name with XX as the second and third characters is encountered; then, the family is processed as a unit. Punch files cannot be grouped in families.

Print Files

Print files undergo some processing before being printed. This processing generates a compressed ASCII file with ANSI carriage control characters, in the following way:

- Two or more blanks (#20) are compressed by replacing them with the ASCII escape control character, ECS (#1B), followed by a count of the number of blanks. The ASCII character 0 (#30) is added to the count of blanks to ensure that the result is beyond the range of ASCII control characters.
- If the internal characteristics field (ic) of the file is ASCII with ANSI carriage control, then the carriage control characters are assumed to be correct and are not looked at by the system.
- If the internal characteristics field (ic) of the file is ASCII with ASCII carriage control, then the carriage control characters are changed to ANSI by the system. The ASCII form feed control character, FF (#0C), when it occurs as the first character of the file after a unit separator, US (#1F), is replaced with the ANSI page eject control character 1 (#31). The ASCII single space, which is a line without the FF after the US, is changed by inserting the ANSI space-one-line control character, blank (#20), after the US.
- The maximum length that is printed at one time is 1+(1-1/2 times the total length) of the file (or family) up to a maximum of #1000 pages. Any file that goes over this limit is printed in parts. If a family of files goes over this limit, the family is divided at the end of a family member. If one file goes over this limit, the file is divided at a random point.
- Family files are concatenated into one file. The file separator, FS (#1C), at the end of all but the last file is replaced with a blank (#20). The start of all files is changed to Hex 31. A maximum of 101 members of a family is printed at one time. Any family that goes over this limit is printed in parts, with the family being divided at the end of a family member. However, the family members that are grouped together are the first 101 members found so that they might be printed out of sequence.

Print Control Characters

Print file control characters can follow either ASCII control character conventions or ANSI conventions.

In the ASCII schema, print control is governed completely by the appearance of ASCII control characters. The FF control character must be the first character of a file or must immediately follow a unit separator (US). The ASCII control characters and their effect on vertical spacing are:

Control Character	Vertical Spacing
FF (#0C)	Page eject
US (#1F)	Single space

In the ANSI print control conventions, the first character of a printer/display output record is not printed or displayed; it is interpreted for vertical spacing control. The first character of each output record directed to the card punch, or any device other than a printer or a display unit, is transmitted and recorded just as any other character in the record without any special action. Characters and their effect on vertical spacing before printing or displaying the next record are:

Character	Hexadecimal Code	Vertical Spacing
blank	#20	Single space
0	#30	Double space
1	#31	Page eject
+	#2B	No vertical advances; move to the first position of the same line
-	#2D	Triple space
other	other	Single space

FILE TERMS

The following paragraphs define terms relating to file allocation and access.

File Extendability

A mass storage file might reside in up to four noncontiguous physical areas called segments. Each segment is a contiguous area; all segments are on the same disk. One or two segments are used when the file is created; additional segments are allowed for extensions. An attempt is made to allocate the initial file size contiguously. If no space large enough is available on any disk, the file is allocated on the first unit having two segments large enough to cover the entire file. The total amount that a file may be extended is a percentage of the creation size of the file. This percentage is an installation parameter, EXTSIZ.

The segment allocation formula for extensions is:

S = (I*EXTSIZ/100-ESA)/NSA

where: S = segment length in small pages; for large pages, S is rounded up to a multiple of 128 small pages.

I = initial file creation length in small pages.

ESA = extension space already allocated in small pages.

NSA = number of remaining segments available to be allocated.

Assume that a file size of 64 pages is created contiguously and the installation parameter EXTSIZ is 50. Substituting these numbers in the segment allocation formula, the extension sizes are:

First extension:

10 = (64*50/100-0)/3, allocate 10 pages.

Second extension:

11 = (64*50/100-10)/2, allocate 11 pages.

Third extension:

11 = (64*50/100-21)/1, allocate 11 pages.

If the calculated space is not contiguously available for small page extensions, the system allocates as much contiguous space as is available. Again assuming initial file size of 64, EXTSIZ of 50, and the largest available contiguous space of 8 pages, the allocation of each extension becomes:

First extension:

10 = (64*50/100-0)/3 — 10 pages needed, but allocate only 8 pages since that is the maximum available.

Second extension:

12 = (64*50/100-8)/2 — 12 pages needed but allocate only 8 pages since that is the maximum available.

Third extension:

16 = (64*50/100-16)/1 — 16 pages needed but allocate only 8 pages since that is the maximum available.

If 12 or 16 pages become available before an extension is needed, the system allocates that amount.

If the calculated space is not contiguously available for large page extensions, the system attempts to allocate the large pages in two segments. In any large page case, the total extension space is a multiple of 128 small pages.

After an extension is performed, the system sends a message to the user informing him of the extension and the new file size. If no space is available or all segments are allocated, the program is aborted with a message.

Files created internally for use by the operating system are contiguous and nonextendable. All files created by existing programs, utilities, and FORTRAN run-time default to extendable files.

File Access Security

Up to eight levels of security, from the lowest level (0) to the highest level (7), can be defined by an installation. The installation can define a security level at or above which the file space will be patterned when the file is destroyed.

Each user is administratively assigned a maximum security level when the user number and account identifier is assigned. Whenever a LOGON is processed, the security level specified by the user is checked against the user's maximum security level. If the level specified is higher than the maximum allowed for that user, system access is denied. The same check is made whenever a user creates or accesses a file. If the security level given is greater than that allowed the user, file access is denied. Although the user satisfies ownership and integrity tests, actual access to the file is further conditioned by the security level.

File Access

File access is controlled by several different parameters in the system. When a file is opened, the access indicates the intended use of the file by the task.

The file ownership is the first access control. All users have access to public files; authorized users have access to pool files; only the owner has access to private files. (The exception is a task running under a privileged user number. Privileged tasks can access any file in the system, except private local files.)

A file having write access can be written into by a user program or the operating system. In the case of data file, this means that modified pages are returned in place to the original file.

An attempt to write explicitly into a read-only file produces an input/output error. An attempt to modify a page from a read-only virtual file produces a fatal error. There is, however, a means for mapping in portions of read-only files and giving those portions write temporary access.

File Management

The possibilities for creation, access, and disposition of a file differ according to whether the file is specified to be a mass storage file, scratch file, write-temporary file, drop file, or output file. These categories are controlled by management category bits in the file index table associated with files on mass storage devices.

Mass Storage Files

Mass storage file creation and disposition is controlled by the originating user. The operating system protects these files from access or destruction by other nonprivileged users.

Scratch Files

Scratch files can be created only by a user program. They exist during the originating task's activity. When a task terminates normally, all scratch files are destroyed unless they are open to other tasks of that user. When the operating system terminates the task or the task terminates and saves its drop file, scratch files are saved. A message to close a scratch file destroys it. All scratch files have read/write access.

The management category scratch should not be confused with local. Local files survive individual task execution and exist for the duration of the user job.

Output Files

Output files contain information suitable for processing to some special output device, such as a printer, card punch, or microfilm device. These files can be created only by a user program. Upon normal task termination, the operating system gives all output files with valid disposition codes to system privileged user tasks for processing to output devices. After files have been processed, they are destroyed. If the disposition code is not valid, the file remains as an unattached permanent file. A message to close an output file with valid disposition also causes the operating system to give the file to a system privileged user task for output processing.

Drop Files

The drop file is a file created by the operating system when a task comes into execution. Each time a control statement calls a task into execution, the operating system creates a drop file for that program. The executing task is known as the source file.

Since the CYBER 200 system requires that each page in memory have some disk correspondence for its current image, cases often occur where a page with a read-only disk image is modified or a free page is assigned, and hence disk space is required which has not previously been specified by the user. To handle this situation, whenever an execution is started, the system automatically provides a file called a drop file into which these pages can be mapped. The actual disk correspondence is kept in the drop file map portion of the first minus page, and two minus pages as well as page zero are entered in the drop file.

The drop file contains modified pages of the source file, free space, and write-temporary files. Modified pages for other files are written directly to the respective file.

The user should be aware that the drop file map is constructed essentially on a page-by-page basis, and the drop file map is of finite size. Attempting to add a page to a drop file map which is full is a fatal error. Users desiring large blocks of virtual space not represented in some file should create a virtual file and map it into the desired space to avoid this difficulty (see volume 2).

The system creates the drop file name from the source file name: the source file name is shifted right two characters, and the new first (leftmost) character created is a number based on the suffix of the logged-on user. (If suffix is A, number is 1; if suffix is B, number is 2, and so forth.) The second character created is the controllee level number of the program, as follows:

1 = Batch processor or virtual system interactive processor
2 = Program initiated by Level 1 program
3 = Program initiated by Level 2 program
4 = Program initiated by Level 3 program
5 = Program initiated by Level 4 program

The length of the drop file is taken from the length specified in the file index table for the file. If zero has been specified there, the length of the drop file is taken from the length specified in the minus page of the source file. If zero has been specified there as well, then the length of the drop file is determined from the source file size and drop file map space. The minimum drop file size is an installation parameter, which is currently #25.

A task can also create its own drop file (see volume 2) which causes the automatic drop file to be destroyed. This can be done only if no pages have been written to the existing drop file. The drop file is preserved for any abnormal termination and can be preserved or destroyed, at the option of the task, upon normal completion.

Write-Temporary Files

Write-temporary files have read-only access when in mass storage. When brought into main memory, such a file can be modified; the modified image becomes a part of the drop file. Subsequent references to the page of the read-only file access the modified page. For the duration of the job, the original source image can be referenced again only by removing the modified image from the drop file. Modified pages are not written back to the file.

TYPES OF INPUT/OUTPUT

CYBER 200 allows two modes of input/output: explicit and implicit.

Explicit input/output involves buffer operations. It provides a conventional manner of data transfer between main memory buffers and mass storage or tape. Explicit input/output offers an advantage over implicit input/output in that more than one page can be transferred between the buffer and a storage device with

a single system request. On the other hand, explicit input/output requires more system action to lock down the buffer and perform other functions, so that implicit input/output is often more efficient for small files. Explicit input/output should be used for large files which transfer many pages at one time.

Implicit input/output does not involve buffers. Information transfers directly to a storage device from its current location in main memory. Implicit input/output should be used to access small files. It should also be used for tasks that access the same part of a file many times.

Implicit input/output is accomplished by the operating system when the user causes an access interrupt by referencing a page of data or code not in memory. If the virtual page has been previously associated with physical space, the system transfers the data between memory and the physical device. If a virtual-to-physical relationship has not occurred previously, the system defines the virtual page in free space so that it becomes an extension of program space. When doing implicit input/output, the virtual address can be considered a symbolic reference to mass storage.

Input and output on mass storage can be implicit or explicit. Implicit input/output is the normal mode of CYBER 200; it uses the file map to associate virtual address space with mass storage space. Obtaining data from a virtual address is an implicit read from a mass storage file; storing data into a virtual address is an implicit write to a mass storage file. (The operating system handles the explicit transfer between main memory and mass storage.)

TYPES OF FILES

CYBER 200 recognizes two types of files: virtual, with a minus page; and physical, without a minus page.

Both virtual and physical files can be accessed implicitly or explicitly. Each uses a system map that associates addresses referenced by a task with a physical address on a storage device, but the entries in the file map differ depending on the selected input/output mode.

When either physical or virtual files are opened in explicit mode, the system makes entries in a table known as the bound explicit map that is part of the drop file. These entries are used by the input/output system routines when explicit input/output requests are issued.

When a file is opened in implicit mode, the mapping information provided by the user is placed in a table known as the bound implicit map. Map entries in the bound implicit map (part of the drop file) relate a set of virtual addresses to a set of mass storage addresses allocated for the file. The maps have an entry for every discontinuity in virtual address space.

Physical Data

A physical file is, by definition, a data file. It cannot be executed. Input/output to or from the file can be performed by the program explicitly or implicitly. A physical file does not have a minus page.

Virtual Code

A virtual code file (controllee file) must be produced by the loader. The first page of the file is known as the minus page; it contains control information used by the operating system and is used during task execution to store such information as the execute and interrupt invisible packages, time-slicing data, input/output connection blocks, maps of defined virtual space, and page fault statistics. The loader creates and initializes the minus page.

The format of a virtual code file is shown in figure 3-3. A detailed explanation of the minus page and other system tables can be found in volume 2.

In many cases, only one bound virtual map entry is created for the controllee file. If discontinuous address spaces or large pages are defined, additional map entries are made. Labeled common is preset to all zeros (64-bit words) and data is loaded as specified. Numbered and blank common areas are mapped into the drop file map. Common blocks that are multiples of 512 are forced to small page boundaries. On the first access to this common area, the user obtains a page created by the operating system and initialized to:

#000C1F1C #000C1F1C

Any modified pages of the controllee file are mapped into the drop file by the operating system.

MASS STORAGE FILE STRUCTURE

Files on mass storage can be unstructured or structured. Both types of files can contain either binary or ASCII data. All files are sequential.

Only the basic unstructured files and System Record Manager structured files are relevant to applications programmers. Within the system, several internal file structures might exist, such as record-structured files or loader files. These are not documented in this reference manual.

Unstructured Files

Unstructured files do not have special control words added internally by the system. They are simply a set of words on mass storage.

From an external standpoint, most unstructured files contain ASCII data and the files themselves are often called ASCII files. An unstructured coded file is a string of ASCII data characters that is terminated by the ASCII control character FS. Each data character is represented by an 8-bit byte in ASCII. Up to three levels of file structure markers can occur within an ASCII data file. The file structure markers can appear in any byte of a machine word and no padding is necessary.

The representation of file structure markers depends on the medium on which the file is stored, as shown in table 3-3. Each separator card and each data card forms an ASCII unit.

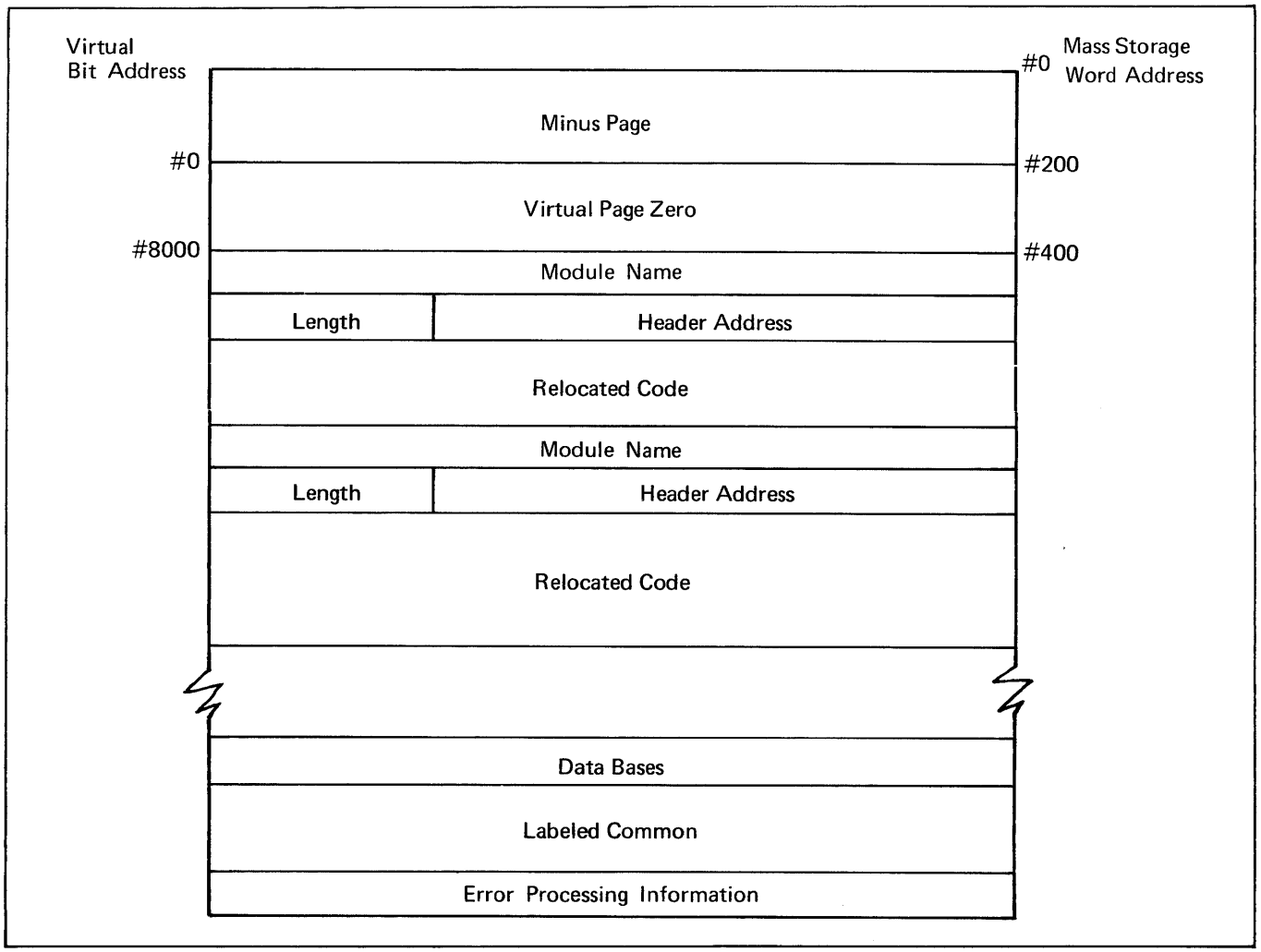


Figure 3-3. Controllee File Format

TABLE 3-3. PUNCH FILES AT UNIT RECORD STATION

File Level	ASCII Control Character	Unstructured File Representation	Card Representation	SRM Structured File Representation	SRM File Level
Unit containing a string of characters	US	#1F	end of card	#1F in record	- -
Record containing one or more units	RS	#1E	7/8/9 card	Control word Code 1	Record
Group containing one or more records	GS	#1D	6/7/9 card	Control word Code 3	Section
File containing one or more groups	FS	#1C	6/7/8/9 card	Control word Code 7	File

On mass storage, two or more consecutive blank characters are compressed to minimize the amount of storage required. Compression is indicated by two bytes: the first byte contains the ASCII control character ESC, which is stored as #1B. The second byte indicates the number of blanks plus #30; the addition of #30 results in a value that cannot be mistaken for any other ASCII control character. Most ASCII files manipulated by the system utilities use blank compression.

Unstructured files can, but need not, be processed by System Record Manager.

System Record Manager Structured Files

System Record Manager structured files have control words created and maintained by the SRM routines. The control words indicate levels within the file and allow records to be named. These files result from FORTRAN statements for unformatted WRITE or BUFFEROUT, and from calls to SRM routines with specified requests for structured files.

Both ASCII data and binary data can appear within a single SRM-structured file as long as each type of data is confined to a single record. The smallest unit for each read or write operation on an SRM-structured file involves all data between control words. As shown in table 3-3, the smallest level of ASCII data recognized by SRM is equivalent to the ASCII control character RS.

Control words provide structural information. In addition, a transparent segmentation scheme is employed by SRM to facilitate file processing. Segmentation of data elements is strictly internal to SRM and has no effect on the logical structure of the file. The control word used for segmentation is called a segment header. A record might be composed of several segments, but the external user need not be concerned with this aspect of structured files. A control word also can be thought of as a link in a two-way linked list in which the link defines the boundary condition at that point in the list, the data mode, and the length of the following data element in the list. Each control word starts on a 64-bit word boundary and has the format shown in figure 3-4. Individual fields of the control word can be obtained by the task with the SRM call GETFIT.

FILE UTILITIES

The CYBER 200 file utilities are used for dumping permanent, pool, or public files from system mass storage to either mass storage or magnetic tape (DUMPF), loading files from the dump devices (LOADPF), and producing printed reports on the status of each permanent file (AUDIT).

These utilities exist as public files and can run in either batch or interactive user mode. When interfacing to tape, the DUMPF and LOADPF routines request the operator to assign an ANSI-labeled magnetic tape. The tape is rewound prior to writing or reading, and it is unloaded at task termination.

The utilities can be run either as privileged or nonprivileged user tasks. As privileged user tasks, they have access to any mass storage file in the system, except local files and attached permanent files. However, as nonprivileged user tasks, DUMPF and LOADPF have access only to the user's attached permanent files. AUDIT has access to public files, pool files for attached pools, and unattached permanent files.

MAGNETIC TAPE FILES

The following material describes tape access and reformatting.

TAPE FILE USE

Tape files are opened only when a controllee calls for one. No analog exists to the REQUEST or LABEL control statements of the NOS or NOS/BE operating systems, although similar processing can be achieved.

Depending on the language of the task, several options are available for accessing a tape file.

A FORTRAN language programmer can select any one of the following:

1. Access an unlabeled tape file through standard FORTRAN language READ and WRITE statements. The PROGRAM statement must specify explicit input/output and tape characteristics, such as UNITn 7,556,1 for a binary 7-track tape. Routines internal to the compiler then generate messages that result in the tape being opened to the program.

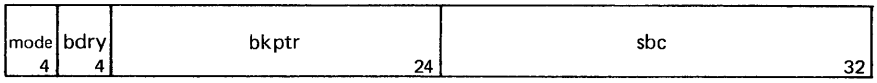
2. Access a labeled or unlabeled tape file through a set of magnetic tape subroutines which can be called through a program. These eight routines attach the program to a tape unit, open a file, and check or write any standard volume and file labels, read or write data, and close the file. They require little programmer knowledge about internal system operations. The routines are accessible through calls to names in the format Q8xxxxTP, as described in section 6. Their use is encouraged for any program that is to become a utility available as a public file.

3. Access unlabeled tape files through calls to System Record Manager routines. Unlike the magnetic tape subroutines, the SRM routines require the programmer to know more about internal system operation, since the first SRM call deals with a file information table that contains fields similar to those in system messages normally accessible only through an assembly language program. They do offer greater programmer control of error processing and flexibility in record processing, however. These calls are described in section 6.

Labels on tapes are supported through SRM and the magnetic tape subroutines. Labels conforming to American National Standard X3.27-1969, Magnetic Tape Labels for Information Interchange can be processed. The magnetic tape subroutines and SRM issue the system messages required to communicate with the station operator regarding mounting of the appropriate tapes. In the absence of calls to either the magnetic tape subroutines or SRM, a program has the responsibility for processing labels with user-written code.

One means of processing a labeled tape without reverting to full programmer responsibility for conversion and operator communication is through a combined use of SRM calls and standard language READ. Once labels are processed through SRM, however, all further file operations should be confined to standard READ access to the file. A programmer should not mix SRM access with standard language access to a file until such time as the programmer is familiar with FORTRAN run-time use of tables internal to SRM.

A. Control word format:



mode — Data mode of segment following this control word:

- 0 Binary
- 1 ASCII
- 2 A name follows (exists only when a 7/8/9 separator card has a name punched in it)
- 4 Another control word follows
- F Beginning-of-information (BOI)

bdry — Boundary code: SRM writes bdry of #C for BOI. SRM recognizes #F to maintain compatibility with earlier versions.

- F Beginning-of-information
- 0 Segment header
- 1 Record separator (RS)
- 3 Group separator (GS)
- 7 File separator (FS)

bkptr — Back pointer: the word index of the preceding control word, if any, relative to the current control word. For a BOI control word, bkptr is set to #FFFFFF to facilitate detection of BOI while backspacing.

sbc — Segment byte count: the number of data bytes in the segment following this control word.

B. Mode=2 control word format:

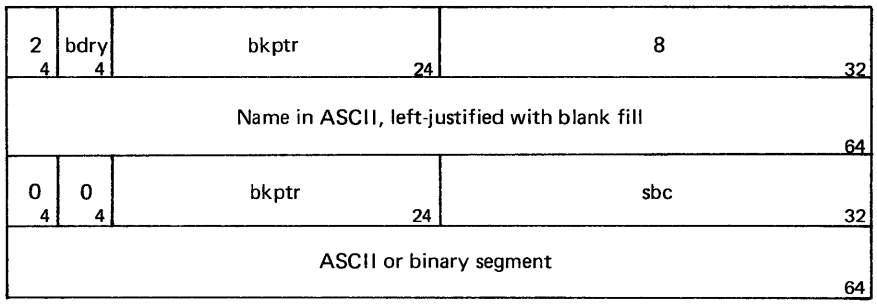


TABLE 3-4. ASCII CODED FILE MARKERS

Assembler language programmers have options similar to FORTRAN programmers.

1. Access a labeled or unlabeled tape using the magnetic tape subroutines also available to the FORTRAN language programmer. Use of these routines is encouraged for any task designed to become a system utility.

2. Access a labeled or unlabeled tape file through System Record Manager.

3. Access a tape through the EXPLICIT I/O message.

Any processing must be performed by explicit, rather than implicit, input/output.

Tape Density

Tape density is affected by the tape drive. Density on 7-track can be:

200 bpi
556 bpi
800 bpi

On 9-track tape density can be:

800 bpi
1600 bpi

The magnetic tape subroutines always read the tape at the density indicated by the tape label, even if it is different from the requested density.

Tape Structure

No logical record structure exists at the tape drive level other than that provided by the interblock gaps and tape marks physically recorded. Record size is not constrained.

Parameters the programmer must specify to describe tape processing depend on the utility or message used to open the tape. These differences are described elsewhere in the operating system reference manual.

Tape mode for 9-track tapes must always be binary even when it is known that the data is a particular character format. For 7-track tapes, either binary or coded can be specified.

TAPE USE BY UTILITIES

The utilities described in this manual cannot, for the most part, be used with magnetic tape files. In particular, the COPY utility and the COMPARE utility cannot reference tape files.

A tape copy utility, TCOPY, is available to copy tape information between mass storage and a tape unit. The utility is also required to copy data from one tape to another.

The utilities DUMPF and LOADPF can be used to dump files to magnetic tape and reload them.

CONTROL STATEMENTS 4

Control statements are executed within a batch job or through an interactive terminal. Table 4-1 lists all control statements by general function and use.

All of the control statements that follow in this section can be executed interactively except the job statement, COMMENT, EXIT, READCC, RERUN, NORERUN, and the TV control statement. These control statements are processed directly by the batch processor. A user file with a name matching one of these control statement names never is executed from within a batch job.

TABLE 4-1. CONTROL STATEMENT FUNCTIONS

Name	Function
Batch Job Only	
COMMENT	Send message to dayfile.
EXIT	Abnormal termination path.
Job Statement	Batch job indicator.
NORERUN	Set norerun status.
READCC	Read alternate control card file.
RERUN	Set rerun status.
TV	Termination value check.
System Access	
STORE	Establish batch access through a card reader (see section 2).
LOGON	Establish interactive access through a terminal (see section 2).
File Management	
ATTACH	Attach permanent files.
AUDIT	List file information.
COMPARE	Compare file contents.
COPY	Copy mass storage file.
DEFINE	Create permanent file or make local file permanent.
DUMPF	Dump files.
FILES	List files.
GIVE	Change file owner.
LOADPF	Reload files.
PURGE	Evict permanent or pool files.
REQUEST	Create local file.
RETURN	Evict local files or detach permanent files.
Pool file utilities	Create, access, or destroy a pool of files that can be accessed by other users.
ROUTE	Specify file disposition.
SWITCH	Change file characteristics.
Debugging	
DEBUG	Symbolic debug (see section 8).
DUMP	Dump drop file (see section 8).
LOOK	Symbolic dump (see section 8).
Tape Manipulation	
TCOPY	Tape copy and file reformat.
File Update	
UPDATE	Card file maintenance (see section 5).
Privileged User Only	
EDITPUB	Add/destroy public file.
Load File	
LOAD	Create controllee file.
OLE	Object library editor.

The functions performed by the utilities described in this section are the same for both types of access.

The format of parameters passed to the utility generally follows the same conventions:

All addresses are assumed to be hexadecimal constants without any special indication.

All other digit strings are assumed to be decimal digits unless preceded by the character #.

Decimal and hexadecimal parameters other than addresses usually, but not always, can be substituted for one another. Substitution cannot occur where decimal or hexadecimal is specifically noted in the parameter descriptions.

BATCH JOB CONTROL STATEMENT FORMAT

All control statements submitted as part of a batch job have the same general format:

task,parameters. comment

Any blanks before the task name are ignored. The task name can be followed by any of the following separator characters, although the comma is shown as a separator in all formats in this manual.

(, blank

If the task name and a parameter are separated by more than one blank, only one blank is passed to the task.

The end of a control statement must be a terminator in the form of a right parenthesis or period. Blanks to the right of the terminator are ignored. If a terminator does not exist on any individual card, the immediately following card is presumed to be a continuation. (A COMMENT control statement cannot be continued.) No special continuation character exists.

Any characters after the terminator are presumed to be a comment. These characters are copied to the dayfile, but are not otherwise processed.

The parameters of a control statement are checked by the utility called, not by the batch processor itself. Any errors in the parameters submitted or any errors encountered during execution of the utility are reported to the dayfile unless otherwise noted. Successful execution is also reported with a status message to the dayfile.

INTERACTIVE UTILITY EXECUTION

The syntax of a control statement entered through the terminal can take the form of either:

Only the utility name, with an optional right parenthesis or period terminator.

Complete control statement with all parameters on one or more lines.

When only the utility name is entered, the utility responds with a prompting message, such as PLEASE SPECIFY PARAMETERS. The prompting message might also include more specific information about appropriate entries, such as a message SPECIFY: FILENAME, LENGTH, OPTIONS. In response, the user should comply with an entry of one parameter or a string of parameters separated by commas. Each entry must be terminated by a NEW-LINE key.

When the control statement is entered on a single line, the task name must be followed by a blank, a comma, or a left parenthesis. Other parameters can be separated by blanks or commas also. Depending on the utility, some parameters have subfields that are separated by the character slash. Any blank immediately adjacent to a parenthesis, comma, slash, or period is ignored.

With the exception of the LOAD utility, the control statement can be entered on more than one line. (No prompting occurs between continued lines.) The character & must be the last character before the NEW-LINE key is pressed. In this circumstance, the next entry line is presumed to be a continuation of the string of characters in the previous entry. Several lines can be concatenated up to a limit of 4096 characters. Both of the following are equivalent:

```
RETURN FILE1,FILE2,FILE3,FILE4

RETURN FILE1,FILE2,FI&
LE3,FILE4
```

Any error in the parameter submitted or any errors encountered during execution of the utility are reported at the terminal. Successful execution is also reported with a status message.

ATTACH (ATTACH PERMANENT FILES)

The ATTACH control statement is used to gain access to an unattached permanent file. The file is attached to the user/suffix combination. A permanent file can be attached only to a single suffix at a time. Local file names and attached permanent file names must all be unique for a particular user/suffix combination. File attributes are not changed with the ATTACH control statement, and no message is produced upon successful completion.

If the ATTACH control statement specifies a list of permanent files to be attached, all files that can be are attached, even if some files in the list cannot be attached. If a permanent file is already attached, the error is nonfatal and no message is produced. The step is terminated with a fatal error if no permanent file lfn exists, if a local file lfn exists at the suffix, or if the permanent file lfn is already attached to a different suffix.

ATTACH control statement format is shown in figure 4-1.

AUDIT (LIST FILE INFORMATION)

The AUDIT control statement lists information relating to the file status of permanent, public, or pool files. Local files cannot be audited. Permanent files need not be attached. Files to be audited can be selected by file name, pack residence, or pool name. Further qualification of files to be audited can be specified by date, time, and type of the last file operation. Attached or unattached status is not reported on the output file.

ATTACH({ lfn-list })
        { *   }

lfn-list	Parameter list of 1 through 16 file names specifying existing permanent files to attach to this suffix. Each file name must be 1 through 8 letters or digits beginning with a letter (except for drop files).
*	Indicator used instead of lfn to attach all permanent files belonging to the user.

Figure 4-1. ATTACH Control Statement Format

A nonprivileged user can audit only public files, permanent files belonging to that user, and files in pools the user is authorized to attach. A privileged user can audit all public, pool, or permanent files in the system.

AUDIT control statement format is shown in figure 4-2. All parameters are optional and can appear in any order.

Parameters work in logical combinations to determine files to be audited. When PN, UN, PF, and POOL all are omitted, only private permanent files associated with the user are audited. When more than one of these parameters is specified, files must meet all criteria specified before they are audited. Similarly, the options specified by the OP parameter operate in combination to select files. A parameter OP=CM, for example, selects files created or modified after a specified date and time; a parameter NCM selects files that have not been created and have not been modified after a specified date and time. The UN and POOL parameters interact as shown in table 4-2. A nonprivileged user cannot specify any user numbers in UN=list form except 0 (for public files) and the user number under which the task is to execute.

Information produced by a full audit is as listed below; a partial audit ends with the DLEN column. If a file exists on storage as more than a single segment, separate lines appear for each segment. Dates appear as month, day, and year; time as a 24-hour clock. All values are decimal unless preceded by #.

AUDIT, PN=pkid-list, PF=lfn-list, UN=userno, POOL=pl-list, OP=opts, DT=mmddyy, TM=hhmm, LO=x, OU=lfn/len/dc.

PN=pkid-list	List of 1 through 16 identifiers of packs, separated by commas, to be searched for files satisfying the other parameters. If the PN parameter is omitted, all active packs are searched.
PF=lfn-list	List of 1 through 128 names, separated by commas, of files to be audited.
UN=userno	Indicator of files to be audited. For a nonprivileged user either or both of the following can be specified: 0 — All public files. userno — User number under which AUDIT is executing. Default. For a privileged user: u-list — List of 1 through 128 user numbers, separated by commas, of files to be audited. User number of zero specifies public files. ALL — Indicator that all files are to be audited.
POOL=pl-list	List of 1 through 128 pools, separated by commas, to which user is attached. The PATTACH control statement must precede use of this parameter.
OP=opts	File characteristics which qualify files selected by UN, PN, POOL, or PF parameters. Options A, C, M, N and X can be specified in any order; commas must not appear between these characters. A — Files accessed since date and time specified. C — Files created since date and time specified. M — Files modified since date and time specified. N — Reverse the meaning of any A, C, or M specified. That is, the appearance of N changes the meaning of A from accessed to not accessed, the meaning of C from created to not created, and the meaning of M from modified to not modified.

Figure 4-2. AUDIT Control Statement Format (Sheet 1 of 2)

	X	Files expired. That is, files whose creation date plus retention period specifies a date preceding the current date.
	If the OP parameter is omitted, default is none of these options.	
DT=mmddyy	Date to modify the A, C, or M option, in format indicating month of year, day of month, and the last two digits of the year.	
	If the DT parameter is omitted, default is the current date.	
TM=hhmm	Time to modify the A, C, or M option, in a format indicating hours and minutes in a 24-hour clock.	
	If the TM parameter is omitted, default is 0000 which is midnight preceding the day specified by the DT parameter.	
LO=x	Indicator of type of audit:	
	F	Full audit.
	P	Partial audit. Default.
OU=lfn/len/dc	File to which audit information is to be written:	
	lfn	Name of file. Must be 1 through 8 letters or digits beginning with a letter. Default is OUTPUT.
	len	Number of small pages in file. When /len is omitted, default is #40.
	dc	Disposition code indicating processing of file:
		PR Print on any available printer at the end of the utility
		When /dc is omitted, the utility does not cause the file to be printed.

Figure 4-2. AUDIT Control Statement Format (Sheet 2 of 2)

TABLE 4-2. INTERACTION OF UN AND PL PARAMETERS FOR AUDIT, DUMPF, AND LOADPF

Files Processed	Privileged User						Nonprivileged User					
	No UN		UN=list		UN=ALL		No UN		UN=list		UN=ALL	
	No PO	PO=list	No PO	PO=list	No PO	PO=list	No PO	PO=list	No PO	PO=list	No PO	PO=list
User private files	X						X		X	X	X	X
Listed user private files (and public files if UN=0)			X	X					†	†		
Listed pool files		X		X				X		X		X
All files regardless of owner (including public and pool files)					X	X						

†Only AUDIT allows listing of the user's private files and the system public files.

Heading	Meaning
NAME	File name
OWNER	Owner user number, public (0), pool name
TYP	File type: virtual code (C), physical data (P)
MC	Management category: private (PERM), scratch (SCRA), output (OUTP), write temporary (WR.T), user-created drop (U.DR), system-created drop (S.DR), batch (BATC)
ACS	Access permission: read (R); write (W)
EXT	File allocation: segmentable (S), extendable (X)
SL	Security level
PACKID	Pack identifier of mass storage file
UU	Logical unit number of device on which pack resides
SADDR	Hexadecimal physical sector address of segment
SLEN	Number of small pages in segment
DLEN	Number of small pages in drop file associated with a virtual code file
FACT	Accounting information
DORG	Creation date; that is, date of origin
TORG	Creation time; that is, time of origin
DOLA	Date of last file access
TLR	Time of last file access
DOLM	Date of last file modification
TOLM	Time of last file modification
EXP	Expiration date; that is, creation date plus retention period

Figure 4-3 shows an example of audit output produced by the control statement:

AUDIT,UN=0,LO=P.

COMMENT (SEND MESSAGE TO DAYFILE)

The COMMENT control statement is valid only within a batch job. It is executed directly by the batch processor. COMMENT causes the accompanying character string to be inserted in the dayfile.

COMMENT control statement format is shown in figure 4-4. No space need appear after the required period; no ending punctuation is needed at the end of the message. Multiple COMMENT control statements are required to send a message longer than the number of columns available on a single card or card image.

COMMENT, message

message	Characters to be sent to the dayfile. Any characters can be specified, but only those available on the line printer should be specified.

Figure 4-4. COMMENT Control Statement Format

COMPARE (COMPARE FILE CONTENTS)

The COMPARE control statement compares the contents of one attached file with the contents of another. If contents do not compare, nonmatching words are written to the dayfile of a batch job or are displayed at an interactive terminal. Both physical and virtual files can be compared. This utility cannot be used with magnetic tape files.

COMPARE control statement format is shown in figure 4-5. The first two parameters are required. All other parameters are optional and can appear in any order.

```
  NAME      OWNER TYP MC   ACS EXT SL PACKID UU  SADDR  SLEN  DLEN

LABEL01         0 P PERM R    SX   0 TPAK01  1 #00140     1
PFI01           0 P PERM R    SX   0 TPAK01  1 #00141    31
BADS9101        0 P PERM RW   SX   0 TPAK01  1 #0801F     1
CETR9101        0 P PERM RW   SX   0 TPAK01  1 #00000   320
CETR9201        0 P PERM RW   SX   0 TPAK01  1 #08020   480

CETR9301        0 P PERM RW   SX   0 TPAK01  1 #0FFA0   320
LABEL02         0 P PERM R    SX   0 TPAK02  2 #00140     1
PFI02           0 P PERM RW   SX   0 TPAK02  2 #00141    31
BADS9102        0 P PERM RW   SX   0 TPAK02  2 #0801F     1
CETR9102        0 P PERM RW   SX   0 TPAK02  2 #00000   320

CETR9202        0 P PERM RW   SX   0 TPAK02  2 #08020   480
```

Figure 4-3. AUDIT Sample Output

COMPARE, alfn, blfn, L=len, A=aadr, B=badr, N=lt.

alfn,blfn	Names of files to be compared.
L=len	Hexadecimal number of words to be compared. If the L parameter is omitted, comparison stops at the end of the shorter file.
A=aadr	Relative hexadecimal word address in file alfn at which comparison is to begin, counting the first word of the file as 0. If the A parameter is omitted, comparison begins with the first word of file alfn.
B=badr	Relative hexadecimal word address in file blfn at which comparison is to begin, counting the first word of the file as 0. If the B parameter is omitted, comparison begins with the first word of file blfn.
N=lt	Decimal number of nonmatching words allowed before comparison stops. Both the nonmatching words and their relative locations are displayed. If the N parameter is omitted, default is 1.

Figure 4-5. COMPARE Control Statement Format

Any compare operation for virtual files should take into consideration that the first 512 words of a virtual file contain the minus page and that the second 512 words of a virtual code file are page zero. The A and B parameters (which must be specified in hexadecimal) can be used to omit these system pages from the comparison.

An example which compares virtual code files FILE1 with FILE2, omitting the minus pages and displaying up to 30 nonmatching words, is:

```
COMPARE,FILE1,FILE2,N=30,A=200,B=200.
```

COPY (COPY MASS STORAGE FILE)

The COPY control statement makes a copy of a mass storage file. Both physical and virtual files can be copied. The utility cannot be used with magnetic tapes.

COPY control statement format is shown in figure 4-6. The first two parameters are required and must be in the order shown. All other parameters are optional and can appear in any order.

The file inlfn must be attached. If file outlfn does not exist or is not attached when COPY is called into execution, the utility creates the file. The new file is a local file with the same characteristics of file inlfn: type (physical, virtual code), management category, security level, internal characteristics (ASCII or binary), and length.

The copy operation terminates when the end of the input file is reached, the end of the output file is reached, or the number of words specified by the L parameter has been copied, whichever occurs first. Status and error

COPY, inlfn, outlfn, L=len, I=inadr, O=outadr.

inlfn	Name of file to be copied.
outlfn	Name of file to contain a copy of all or part of file inlfn. Can be either an existing file or a new file to be created by the utility.
L=len	Hexadecimal number of words to be copied. If the L parameter is omitted, file inlfn is copied from either the file beginning or from any inadr specified.
I=inadr	Relative hexadecimal word address in file inlfn where copying is to begin, counting the first word of the file as 0. If the I parameter is omitted, inlfn is copied from its beginning.
O=outadr	Relative hexadecimal word address in file outlfn at which copied information is to be placed, counting the first word of the file as 0. If the O parameter is omitted, copy begins at the beginning of outlfn.

Figure 4-6. COPY Control Statement Format

information from the utility is returned to the dayfile of a batch job or to the terminal of an interactive user. The hexadecimal number of words copied is displayed.

When the length of the information to be copied is less than the full input file length, or when a second copy operation is to add information to the output file, the COPY control statement should be preceded by a DEFINE or REQUEST control statement to establish an appropriate length for the output file. For example, assume file INLFN exists as a physical data file of 20 small pages in which only the first page and the last page are to be copied. Also assume OUTLFN does not exist. Appropriate control statements to copy to file OUTLFN are:

```
DEFINE,OUTLFN/2,T=P.
COPY,INLFN,OUTLFN,L=200.
COPY,INLFN,OUTLFN,L=200,I=2600,O=200.
```

In the absence of DEFINE, OUTLFN is established with a length of one page, as specified by the L parameter. The second COPY would terminate immediately since the output file is full.

The first 512 words of any virtual file contain the minus page the system uses to equate virtual addresses with actual mass storage addresses. If the minus page is not to be copied or overwritten, the I and O parameters (which require hexadecimal values) must be used. I=200 and O=200, for example, skip the minus pages. Similarly, the second 512 words of a virtual code file contain page zero for the file; if the zero page is not to be copied, I should be further adjusted to I=400.

DEFINE (CREATE PERMANENT FILE OR MAKE LOCAL FILE PERMANENT)

The DEFINE control statement defines a permanent file. DEFINE can be used to create a permanent file or to make a local file permanent. DEFINE must be used to ensure that a file is created on a particular pack. It can also be used to determine whether a given pack has adequate space to hold a file of the required size; if not, the utility returns a fatal error code. This utility controls whether mass storage allocated to the file is contiguous at creation and whether the file can be extended.

Execution of DEFINE, either to create a permanent file or make a local file permanent, results in appropriate entries in the pack file index. Creation of a permanent file results in allocation of mass storage.

The NOSEGMENT and NOEXTEND parameters are used to control the continuity of the file. The interaction between the NOSEGMENT and NOEXTEND parameters is as follows:

Extendable File	Segmentable File	Result
No	No	One segment. File cannot be extended.
No	Yes	File created as one or two segments. File cannot be extended.
Yes	No	File created as one segment. Noncontiguous segments can be added.
Yes	Yes	File created as one or two segments. Noncontiguous segments can be added.

DEFINE control statement format is shown in figure 4-7. The first parameter must be the file name. File length, if specified, must be the second parameter. All other parameters are optional and can appear in any order. If a

DEFINE, lfn/len, ACCESS=acs, TYPE=typ, SECURITY=lvl, PACK=packid, NOEXTEND, NOSEGMENT,.

lfn	Name of mass storage file to be created. Must be 1 through 8 letters or digits beginning with a letter (except for local drop files, which can be made permanent).
/len	Number of small pages to be allocated for the file stated as a decimal or hexadecimal number 1 through #FFFF. If this parameter is omitted, default is installation defined.
ACCESS=acs	File access permission: R Read. W Write. R and W can be combined in any order without commas. If the ACCESS parameter is omitted, default is RW.
TYPE=typ	File type: C Virtual code file. P Physical data file. Default.
SECURITY=lvl	Security level of file being created. Must be in the range of 1 through 255. If the SECURITY parameter is omitted, default is the level resulting from STORE or LOGON use.
PACK=packid	Identifier for pack on which file is to be created. Pack identifiers are six characters in length: fewer characters are blank padded on the right, while a character string longer than six characters is truncated. If the PACK parameter is omitted, default is a pack selected by the system.
NOEXTEND	Indicator that the file cannot be extended. That is, the file size established by the length parameter is an absolute size. If the NOEXTEND parameter is omitted, the file can be increased to an additional percentage of original file length, as determined by an installation parameter.
NOSEGMENT	Indicator that file space on mass storage must be contiguous when the file is created. If the NOSEGMENT parameter is omitted, the system might allocate initial file space in two segments.

Figure 4-7. DEFINE Control Statement Format

local file is made permanent, length and all other parameters are ignored.

Upon successful completion, the message CREATED PERMANENT FILE or EXISTING LOCAL FILE MADE PERMANENT will be printed.

Retention period for the file is an installation option. The SWITCH control statement can be used to specify a particular number of days the file is to be retained on mass storage.

DUMPF (DUMP FILES)

The DUMPF control statement dumps public, permanent, and pool files to another pack or to a magnetic tape. At programmer option, the original mass storage file is purged when a file is dumped. Files to be dumped can be selected by user number, pack, residence, file name, or pool name. Selection of files operates by a combination of parameters specified. Further qualification of files to be dumped can be specified by date, time, and type of the last file operation.

A nonprivileged user can dump only attached permanent files or attached pool files. A privileged user can dump all permanent, pool, or public files in the system, with the exception of attached permanent files. Local files are not processed. At the completion of a DUMPF, the dump files created from the original mass storage files become unattached permanent files.

When dumping a file to mass storage, DUMPF maintains a directory file for each user. The directory contains the generated file names of all the dumped files for this user on the mass storage device. The dumped file exists as a private file generated by DUMPF. The contents of this file consists of the PFI image as the first block followed by the contents of the file. The format of the directory file and each dumped file are described in figure 4-8.

TPAKMMNN is referred to as the pseudo file. TPAKMM represents the pack identifier as specified by the VSN parameter. NN is the file sequence number which is in hexadecimal. Currently a nonprivileged user can dump #FF files, while a privileged user can dump #3FF files. When the sequence number is greater than #FF, the pseudo file is then represented by PAKMMNNN.

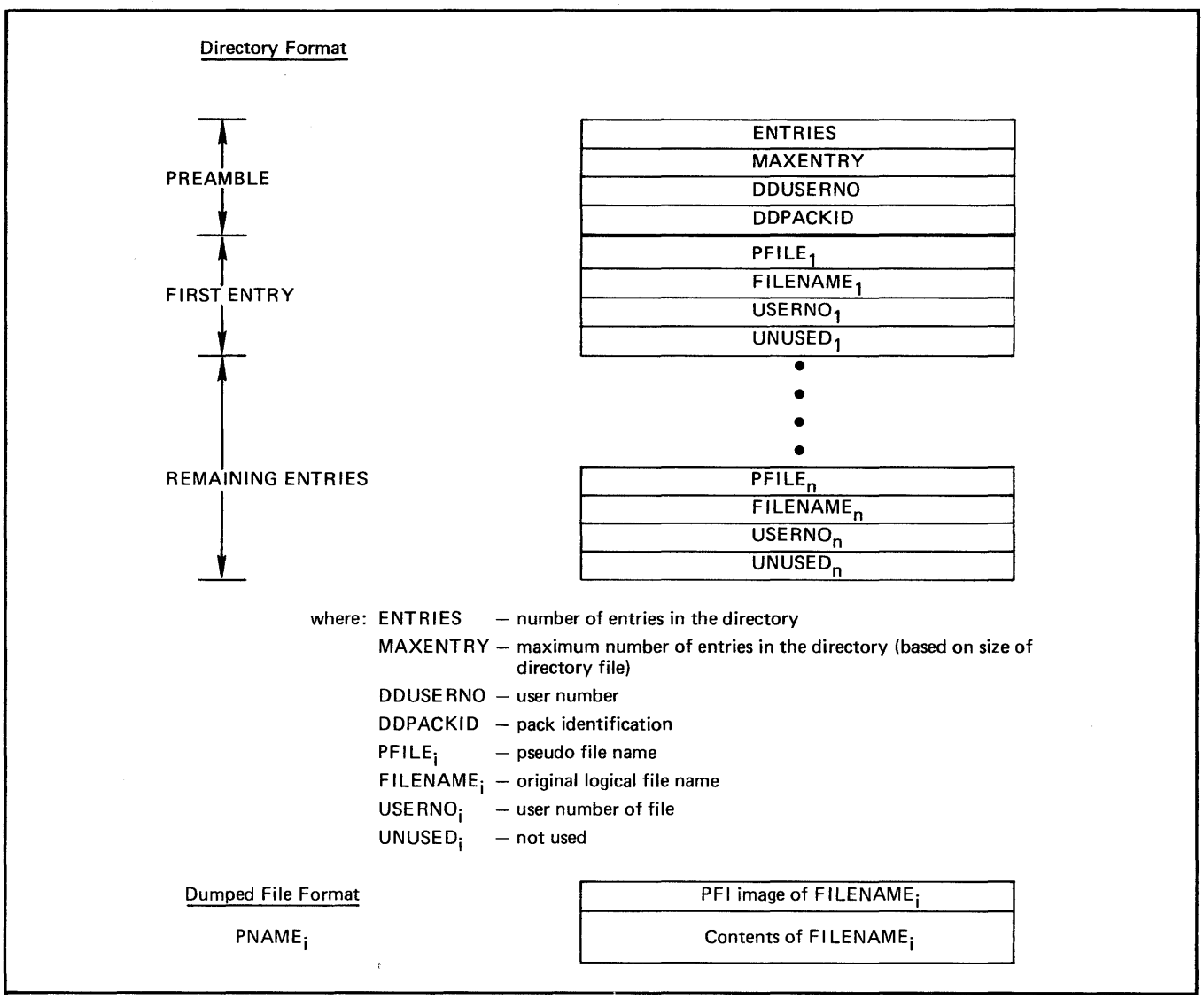


Figure 4-8. Directory/Dumped File Format

When dumping files to mass storage, an entry is created in the directory for each file, LFN. If the file, LFN, already exists in the directory for this user number, the existing LFN is destroyed and the current file, LFN, is created.

DUMPF control statement format is shown in figure 4-9. All parameters are optional and can appear in any order.

Parameters work in logical combinations to determine files to be dumped. When PN, UN, PF, and POOL all are omitted, only private files associated with the user are dumped. When more than one of these parameters is specified, a file must meet all criteria specified before it is dumped. Similarly, the options specified by the OP parameter operate in combination to select files. A parameter OP=CM, for example, selects files created or modified after a specified date and time; a parameter OP=NCM selects files that have not been created and have not been modified after a specified date and time. The UN and POOL parameters interact as shown in table 4-2.

DUMPF, DD=device, VSN=id-list, DE=density, RE=days, UN=userno, POOL=pl-list, PN=pkid-list, PF=lfn-list, OP=opts, DT=mmddyy, TM=hhmm, LO=x, OU=lfn/len/dc.

DD=device — Dump device:

NT	9-track tape.
MT	7-track tape.
MS	Pack indicated by VSN parameter.

If the DD parameter is omitted, default is an installation option.

VSN=id-list — Identification of tape or pack to receive dump:

For tapes, a list of 1 through 128 volume serial numbers of tapes. If omitted, the operator assigns a tape.

For packs, a list of pack identifiers of 1 through 6 characters. Required.

If sufficient tapes or packs are not specified, the operator is instructed to assign additional devices.

DE=density — Density of dump tape:

LO	200 bpi (7-track tape only).
HI	556 bpi (7-track or 9-track).
HY	800 bpi (7-track or 9-track).
PE	1600 bpi (9-track only).

If the DE parameter is omitted, default is an installation option.

RE=days — Tape retention period. 1 through 999 days.

If the RE parameter is omitted, default is an installation option.

UN=userno — Indicator of files to be dumped:

For a nonprivileged user:

userno	User number under which DUMPF is executing.

For a privileged user:

u-list	List of 1 through 16 user numbers, separated by commas, of private files to be dumped.
ALL	Indicator that all private, pool, and public files are to be dumped.

POOL=pl-list — List of 1 through 128 pools, separated by commas, of pools to be dumped.

PN=pkid-list — List of 1 through 16 identifiers, separated by commas, of packs to be dumped.

If the PN parameter is omitted, all active packs are searched for files meeting other dump criteria.

PF=lfn-list — List of 1 through 128 names, separated by commas, of private or pool files to be dumped.

Figure 4-9. DUMPF Control Statement Format (Sheet 1 of 2)

OP=opts	File characteristics which qualify files selected by UN, PN, POOL, or PF parameters. Options A, C, M, N, X, or P can be specified in any order; commas must not appear between these characters.	
	A	Files accessed since date and time specified.
	C	Files created since date and time specified.
	M	Files modified since date and time specified.
	N	Reverse the meaning of any A, C, or M specified. That is, the appearance of N changes the meaning of A from accessed to not accessed, the meaning of C from created to not created, and the meaning of M from modified to not modified.
	X	Files expired. That is, files whose creation date plus retention period specifies a date preceding the current date.
	P	Indicator that file is to be purged from mass storage after it is dumped successfully.
	If the OP parameter is omitted, default is none of these options.	
DT=mmddyy	Date to modify the A, C, or M option, in format indicating month of year, day of month, and the last two digits of the year. If the DT parameter is omitted, default is the current date.	
TM=hhmm	Time to modify the A, C, or M option, in a format indicating hours and minutes in a 24-hour clock. If the TM parameter is omitted, default is 0000 which is midnight preceding the day specified by the DT parameter.	
LO=x	Indicator of type of audit information to be written.	
	F	Full information.
	P	Partial information. Default.
OU=lfn/len/dc	File to which dump status information is to be written.	
	lfn	Name of file. Must be 1 through 8 letter or digits beginning with a letter. Default is OUTPUT.
	len	Number of small pages in file. When /len is omitted, default is #40.
	dc	Disposition code indicating processing of file: PR Print on any available printer at the end of the utility When /dc is omitted, the utility does not cause the file to be printed.

Figure 4-9. DUMPF Control Statement Format (Sheet 2 of 2)

A dump tape or file produced by a nonprivileged DUMPF contains a copy of the formatted PFI prior to opening each file being dumped; therefore, the access fields are not updated on the dump tape/file but are updated on the file index maintained by the system. An attempt to reload the files by date and time of last access uses the original values for those fields for the comparison.

A dump tape or file produced by a privileged DUMPF contains a copy of the formatted PFI after opening each file being dumped; therefore, the access fields are updated on both the dump tape/file and the file index maintained by the system. An attempt to reload the files by date and time of last access uses the updated values for those fields for the comparison.

Pool files can be dumped by any user authorized to access the pool when the pool parameter is specified. Only the pool boss can execute with OP=P to purge the mass storage copy of the file. A PATTACH control statement must precede use of the pool parameter for a nonprivileged user dump of pool files.

DUMPF can execute concurrently with other tasks, including other DUMPF tasks. If a file cannot be dumped, the utility writes an appropriate message for the file specified by the OU parameter and continues with the dump of other files selected. Attached or unattached status is not reported on the output file.

When files are dumped to tape, an ANSI-labeled tape is produced. The DUMPF utility displays appropriate messages for operator action.

EDITPUB (ADD/DESTROY PUBLIC FILE)

The EDITPUB control statement is valid only for privileged user numbers. It adds or destroys a file from the ownership category of public. Files processed with EDITPUB must be attached.

EDITPUB format is shown in figure 4-10.

EDITPUB, {L / D=lfn-list}, N=lfn-list, P=lfn-list, VRI=index.

L	For an interactive call only, indicator that files to be destroyed are to be specified interactively.
D=lfn-list	List of 1 through 16 names, separated by commas, of public files to be destroyed.
N=lfn-list	List of 1 through 16 names, separated by commas, of files to be added to the public file ownership category without privileged status.
P=lfn-list	List of 1 through 16 names, separated by commas, of files to be added to the public file ownership category with privileged status.
VRI=index	Index into the Variable Rate Table in the range 1 through 255, for public files being added with the call. Default is 0.

Figure 4-10. EDITPUB Control Statement Format

When the utility is called with the L parameter from an interactive terminal, it displays the name of each public file in turn and waits for one of the following terminal user responses:

D	File is to be destroyed
NEW-LINE key	File is to be retained
STOP	Utility is to be terminated

Files cannot be destroyed while they are open to any task. If the system cannot destroy a file, it returns error messages to the dayfile of a batch job or to the terminal of an interactive user.

If a file being made public has the same name as an existing public file (that is, if a file is being replaced), the destroy and add operations can be performed through a single call. For example, to replace public files X and Y, the following control statement is appropriate:

```
EDITPUB(D=X,Y,N=X,Y)
```

If the VRI parameter is used, at least one of the N or P parameters must be used. In this case, all files being made public in this control statement must be controllees, and the VRI parameter will apply to all. Files made public using the VRI parameter will not retain read or write access.

If both the L and VRI parameters are used from an interactive terminal, and the user response indicates that any file is to be retained, the VRI for that file is not reset to the VRI parameter value. The VRI file index entry for any file is 0 until modified by a VRI specification in EDITPUB.

EXIT (ABNORMAL TERMINATION PATH)

The EXIT control statement is valid only in a batch job. It is executed directly by the batch processor. It establishes a control path to be followed in the event of abnormal job termination.

Whenever a termination value test fails (see TV control statement), the batch processor searches subsequent statements and resumes execution at the control statement following the first EXIT encountered. If no EXIT control statement exists, the job ends abnormally. If an EXIT statement is encountered during normal job advancement to the next control statement, the job ends normally.

EXIT control statement format is shown in figure 4-11. More than one EXIT control statement can appear in a job.

```
EXIT.
```

Figure 4-11. EXIT Control Statement Format

The termination value is set to 255 when an EXIT control statement establishes the execution path.

If control transfers to the path established by an EXIT control statement because the job time limit is reached, a short amount of time is made available to the job. The batch processor uses the time, however, so that user job tasks might not be performed in this instance.

FILES (LIST FILES)

The FILES control statement lists status information about specified public files, private files, and pool files. Private files include local files at the suffix only and all attached or unattached permanent files.

FILES control statement format is as shown in figure 4-12. All parameters are optional, and, with the exception of a list of private files, can appear in any order. Omission of all parameters is equivalent to FILES (PRIVATE=ALL).

All specified files are listed in alphabetical order. Appropriate headings identify different types of information:

Heading	Meaning
NAME	File name.
D	Duplicate file name flag. An asterisk in this column indicates that at least one other file exists with the same name and owner.
LEN	Actual length of each segment of the file, in decimal number of small pages. Total file length is the sum of all segments shown.

FILES, lfn-list, PUBLIC=lfn-list, PRIVATE=lfn-list, POOL=poolname,lfn-list, L=lfn.

lfn-list	List of files to have status information written.
PUBLIC=lfn-list	Public files to be listed:
	lfn-list List of 1 through 255 public file names, separated by commas.
PRIVATE=lfn-list	Private files to be listed:
	lfn-list List of 1 through 255 private file names, separated by commas.
POOL=poolname,lfn-list	Pool files to be listed:
	poolname Name of pool to which user is attached.
	lfn-list List of 1 through 255 file names in pool poolname. If omitted, all files in pool are listed.
	The POOL parameter can be repeated within the parameter list.
L=lfn	Name of file to which output is to be written. Must be 1 through 8 letters or digits beginning with a letter. Any existing file with the same name is destroyed.
	Default in batch mode is OUTPUT. Default in interactive mode returns output to the terminal.

Figure 4-12. FILES Control Statement Format

S.ADDR	Hexadecimal physical sector address of the segment.
UN	Logical unit number of the device on which the segment resides.
AC	Access permission: read (R), write (W).
TYP	File type: virtual code (C), physical data (P).
MCAT	File management category: disk (DISK), scratch (SCRA), output (OUTP), write temporary (WR.T), user-generated drop file (U.DR), system-generated drop file (S.DR), batch input (BATC).
ORI.DATE	Date file was created: origin date.
RETN	Number of days file is to be retained.
OWNER	File ownership: public (*PUBLIC), permanent (*DISK), local (*LOCAL), pool (poolname). For permanent files, the attached suffix (if any) is designated by a letter, A through D.

The same file name can be specified more than once and might appear in the output more than once, since file names need be unique only in relation to all other files with the same ownership. On output, files with duplicated names are listed in the order: local, permanent, pool, and public.

From an interactive terminal, this utility can be called by name alone and the programmer will be prompted for parameters. Information is displayed 15 lines at a time. When output exceeds display size, the programmer is required to enter CONTINUE to continue the display or enter END to terminate the display.

Figure 4-13 shows an example of output produced by a batch job containing a control statement:

```
FILES, PUBLIC=FORTRAN,CARDREDR,
    PRIVATE=ALL, POOL=POOLXXXX,
    AFILE,POOL=BASELINE,CARDREDR.
```

GIVE (CHANGE FILE OWNER)

The GIVE control statement changes the owner of a local or attached permanent file. Any file referenced must not be open at the time this utility executes.

The current file owner can give one or more files to:

Another user number. In this instance the file ownership category remains private, but the user number changes. The file becomes an unattached permanent file belonging to the new owner.

A pool, if the current owner is the pool boss. In this instance the file ownership category changes to pool. No user, including the pool boss, can then access the file without attaching to the pool through the PATTACH utility.

Once a file is referenced by GIVE, it is no longer accessible through the old user number.

NAME	D	LEN	S.ADDR	UN	AC	TYP	MCAT	ORIGDATE	RETN	OWNER
12BUCAL		00265	#01D01	05	R	C	U.DR	05/10/72	0100	*DISK A
42FORTRA		00140	#01100	01	R	C	U.DR	02/04/77	0005	*DISK D
AFILE		00030	#00432	04	R	C	DISK	01/01/77	1023	POOLXXXX
BATCHJOB		00001	#01111	04	R	P	BATC	02/05/77	0000	*LOCAL
CARDREDR		00035	#01987	03	R	C	DISK	09/09/76	0090	*DISK
CARDREDR		00032	#01FFF	02	R	C	DISK	01/03/77	1000	BASELINE
CARDREDR		00032	#01266	01	R	C	DISK	09/01/75	1001	*PUBLIC
EXTENDED		00020	#03000	05	R	C	DISK	12/31/76	0020	*DISK
		00100	#02C40	05						
		00028	#01129	05						
FORTRAN		00040	#00608	01	R	C	DISK	06/12/73	0200	*PUBLIC
OUTPUT		00512	#00FFF	02	RW	P	OUTP	02/05/77	0000	*LOCAL
OUTPUT		00512	#01864	01	RW	P	OUTP	01/01/77	0005	*DISK A
PXX12345		00200	#01800	02	RW	P	WR.T	02/05/77	0000	*LOCAL

Figure 4-13. FILES Sample Output

GIVE format is as shown in figure 4-14. Parameters can appear in any order.

Public files cannot be referenced by GIVE. Further, any file name referenced by GIVE cannot have the same name as a public file, even though public files are a separate ownership category.

GIVE, { =ALL / lfn-list } , { U=newown / POOL=poolname,SHARE=perm }

=ALL	Indicator that all attached private files associated with the user are to change ownership.
lfn-list	List of 1 through 16 files names, separated by commas, of files whose ownership is to change. Must not include private files with the same name as a public file.
U=newown	Used number of new owner of private files.
POOL=poolname	Name of existing pool to receive files listed.
SHARE=perm	Access allowed for files being given to a pool:
R	Any user attaching to pool can read or execute the file. Default.
W	Any user attaching to pool can write or execute the file.
RW	Any user attaching to pool can read, write, or execute the file.
NONE	No access possible.

Figure 4-14. GIVE Control Statement Format

JOB STATEMENT (BATCH JOB INDICATOR)

A job statement is required as the first statement of each batch job. It must be preceded by a STORE card or a card with 6/7/9 multipunched in column 1 which separates it from a preceding job in the same batch deck. The job statement names the job, establishes the time limit, and determines whether a job continues when errors are encountered.

Job identification statement format is as shown in figure 4-15. The first parameter is required. All other parameters are optional and can appear in any order. Commas or blanks can be used as separators.

jobname, Tt, TVvalue.

jobname	Name of job. 1 through 8 letters or digits beginning with a letter.
t	Decimal number specifying the maximum amount of time a job can run. Time is an installation defined System Time Unit (STU). Default is 1 STU; maximum is 9999 STUs.
value	Initial termination value, decimal 0 through 255, as defined for the TV control statement. Release default is 8, but can be changed by an installation.

Figure 4-15. Job Statement Format

LOAD (CREATE CONTROLLEE FILE)

The LOAD control statement transforms object modules into a virtual code controllee file suitable for execution. The files input to the loader must contain object modules. The files themselves can be either a modmerge file produced by OLE or a file containing a module in the format produced by the CYBER 200 assembler or compiler. The input files must be attached, and the output files are local.

LOAD, lfn-list, CNTROLEE=lfn/len, CDF=dlen, OUTPUT=lfn/len, LIBRARY=lib-list, EQUATE=sub,nam, ENTRY=ept,

DEBUG=mod-list, VR=string, ORIGIN=bitadr, GRSP=list,bitadr, GRLP=list,bitadr, GROS=com-list,bitadr,

GROL=com-list,bitadr, GRLPALL=△, DSA=bitadr.

lfn-list	List of 1 through 10 names, separated by commas, of files containing object modules in format produced by a compiler or assembler or containing a modmerge file produced by OLE. Files are loaded in the order specified. If the lfn parameter is omitted, file BINARY is assumed.
CNTROLEE=lfn/len	Name of file to which the relocatable virtual code is to be written. Must be 1 through 8 letters or digits beginning with a letter. len — Number of small pages in file. Default is #102. The file length is reduced at job termination. If the CNTROLEE parameter is omitted, the default is GO/#102.
CDF=dlen	Number of small pages in drop file to be created at execution time. (The value dlen is stored in word #20 of the minus page.) If the CDF parameter is omitted, the operating system calculates size when the task is executed (and word #20 of the minus page is set to zero).
OUTPUT=lfn/len	Load map file: lfn — Name of file to which load map is to be written. len — Number of small pages in file. Default is #25. Any length specified is reduced by the loader at the end of map construction. If the OUTPUT parameter is omitted, default is OUTPUT/#25.
LIBRARY=lib-list	List of files containing libraries created by OLE from which externals are to be satisfied. If the LIBRARY parameter is omitted, only the file SYSLIB is searched for unsatisfied externals.
EQUATE=sub,nam	List of pairs of external references to be substituted during linking: sub — External name to be substituted for the paired external name. nam — External name to be replaced. Any common names must be preceded by an asterisk to identify them as other than a module name. Blank common is indicated by an asterisk alone.
ENTRY=ept	Name of entry point in a loaded module at which execution is to begin. If the ENTRY parameter is omitted, the eight characters MAIN.ƀƀƀ become the transfer address.
DEBUG=mod-list	List of modules for which the debug version is to be loaded.
VR=string	String of 1 through 8 ASCII characters that are to be stored left-justified and blank filled in register #A. The characters , .) and blank cannot be included in the string. If the VR parameter is omitted, register #A is blank filled.
ORIGIN=bitadr	Virtual bit address at which loading is to begin. The loader adjusts this address upward to a page boundary if necessary. If the ORIGIN parameter is omitted, loading begins at address #8000.

Figure 4-16. LOAD Control Statement Format (Sheet 1 of 2)

GRSP=list,bitadr	Indicator that modules or common blocks are to be loaded as groups beginning at small page boundaries.
	list — List of module names or list of common block names. Common block names must be prefixed by an asterisk. An asterisk alone indicates blank common.
	bitadr — Bit address of small page. Default is next available small page.
GRLP=list,bitadr	Indicator that modules or common blocks are to be loaded as groups beginning at large page boundaries.
	list — List of module names or list of common block names. Common block names must be prefixed by an asterisk. An asterisk alone indicates blank common.
	bitadr — Bit address of large page. Default is next available large page.
GROS=com-list,bitadr	Indicator that common blocks are to be grouped and relocated on small page boundary, but space is not reserved in the controllee file.
	com-list — List of common blocks.
	bitadr — Bit address of small page. Default is after previously loaded items.
GROL=com-list,bitadr	Indicator that common blocks are to be grouped and relocated on large page boundary, but space is not reserved in the controllee file.
	com-list — List of common blocks.
	bitadr — Bir address of large page. Default is after previously loaded items.
GRLPALL=Δ	Indicator that all code, data base, and labeled common is to be grouped on large pages. A blank must follow the = character.
DSA=bitadr	Virtual address at which the dynamic stack begins. Must be a page boundary.
	If the DSA parameter is omitted, default is after last virtual address allocated.

Figure 4-16. LOAD Control Statement Format (Sheet 2 of 2)

LOAD control statement format is shown in figure 4-16. All parameters are optional, although subparameters cannot be separated. Any list of files to be loaded must appear first; otherwise parameters can appear in any order. Any number of items can appear in the lists associated with the LIBRARY, EQUATE, DEBUG, and the GRxx parameters. Multiple GRxx parameters can appear.

If the loader is used interactively from a terminal, the user must type SPACE and NEWLINE to indicate no options or when terminating options.

The number of optional loader files cannot exceed 13. Optional loader files are SYSLIB, files specified in library format, and user files. The number of user files cannot exceed 10. The size of the controllee file is reduced at job termination.

The loading process loads modules from files in the order they are listed on the LOAD control statement. Then, unsatisfied externals referenced by those modules are satisfied from any libraries specified by a LIBRARY parameter. If no library is specified, or if unsatisfied externals remain after a search of all specified libraries, a file with the name SYSLIB is searched for module names that would satisfy externals.

At programmer option, the loader links external references to routines on a library that contain user-written debugging versions of specified routines. Use of the DEBUG parameter changes the linkage, so that a reference to NAME, for example, is linked to debugging routine NAMEQ. Any reference to NAME within NAMEQ is linked to NAME, although other references to NAME are linked to NAMEQ. Both versions of a routine must exist on the same library.

Changes in linkage can also be controlled through the EQUATE parameter. This option allows substitution of external references so that either external entry point names or common area names can be changed at load time.

Loading begins at the virtual address specified by the ORIGIN parameter, which is adjusted upward if necessary to a page boundary. The location at which routines or labeled common blocks are loaded can be controlled by the programmer through the GRSP and GRLP parameters; routines or labeled common that are to have addresses assigned, but are not to be assigned space as part of the controllee file, are specified by the GROS and GROL parameters. Note that although these options provide a grouping capability, no ordering is implied. When using a group option with the BITADR parameter, it is the user's responsibility to prevent using an address that is already allocated.

The loader produces a load map showing the locations of routines and data in the program. The map is written to the file specified in the OUTPUT parameter with ASCII carriage control characters suitable for printing.

The VR parameter aids the programmer in managing the development and usage of a program by providing an identification which is recognizable in a dump. The character string specified by this parameter is stored in register #A of the zero page. The loader also stores the date and time of controllee file creation in registers #B and #C. Date format is the eight characters mm/dd/yy representing the month, day of month, and last two digits of the year. Time format is the eight characters hh.mm.ss representing the hour on a 24-hour clock, minutes of the hour, and seconds.

The dynamic stack address referenced by the DSA parameter is concerned with temporary working space available in the register file. One typical use of the dynamic stack is for saving registers over a call. They are sometimes referred to as a data base. The dynamic stack address is defined as the current stack pointer, starting at the DSA address; the dynamic stack pointer is #180 bits (six words) after the current stack pointer. If the current stack pointer is at #200000, the dynamic stack pointer is at #200180. The loader assigns a default DSA following the last virtual address allocated. The dynamic stack address is always printed on load maps.

LOADPF (RELOAD FILES)

The LOADPF control statement reloads mass storage files that have been dumped to a tape or a pack. Files to be loaded can be selected by permanent file name or pool file name. Further qualification of files to be reloaded can be specified by date, time, or last access.

A nonprivileged user can load only files owned by the user. Pseudo files (see DUMPF for a description of pseudo files) are attached and then returned by LOADPF. Files loaded from mass storage become unattached permanent files. Local files are not processed. A privileged user can load all files in the system. Pool files can be reloaded by any user. Only the pool boss or privileged user can restore file status as a member of a particular pool, however. If the system cannot restore pool status because the pool no longer exists or another file exists with that name, the reloaded file remains a permanent file.

LOADPF control statement format is shown in figure 4-17. The dump device must be adequately specified by DE and DD. All other parameters are optional and can appear in any order.

Parameters work in logical combinations to determine files to be loaded. When PN, UN, PF, and POOL all are omitted, only permanent files associated with the user are loaded. When more than one of these parameters is

LOADPF, DD=device, VSN=id-list, DE=density, UN=userno, POOL=pl-list, PF=lfn-list, OP=opts, DT=mmddyy, TM=hhmm, PN=pkid-list, LO=x, OU=lfn/len/dc.

DD=device	Device on which dump files exist:	
	NT	9-track tape
	MT	7-track tape
	MS	Pack indicated by VSN parameter
	If the DD parameter is omitted, default is an installation option.	
VSN=id-list	List of 1 through 128 pack identifiers separated by commas, on which dump files exist; or of volume serial numbers of tapes. Required parameter.	
DE=density	Density of dump tape:	
	LO	200 bpi
	HI	556 bpi
	HY	800 bpi
	PE	1600 bpi
UN=userno	Indicator of files to be reloaded:	
	For a nonprivileged user:	
	userno	User number under which LOADPF is executing.
	For a privileged user:	
	u-list	List of 1 through 128 user numbers, separated by commas, of files to be reloaded.
	ALL	Indicator that all files are to be reloaded.
POOL=pl-list	List of 1 through 128 pool names, separated by commas, of pools whose files are to be reloaded.	

Figure 4-17. LOADPF Control Statement Format (Sheet 1 of 2)

PF=lfn-list	List of 1 through 128 file names, separated by commas, of files to be reloaded.
OP=opts	File characteristics which qualify files selected by UN, PN, POOL, or PF parameters. Options A, C, M, N, X, and R can be specified in any order; commas must not appear between these characters. A — Files accessed since date and time specified. C — Files created since date and time specified. M — Files modified since date and time specified. N — Reverse the meaning of any A, C, or M specified. That is, the appearance of N changes the meaning of A from accessed to not accessed, the meaning of C from created to not created, and the meaning of M from modified to not modified. X — Files expired. That is, files whose creation date plus retention period specifies a date preceding the current date. R — Indicator that existing file with the same name as a file being reloaded is to be destroyed and the dumped file is to take its place. If omitted, a file with a duplicated name is not to be reloaded. If the OP parameter is omitted, default is none of these options.
DT=mmddyy	Date to modify the A, C, or M option, in format indicating month of year, day of month, and the last two digits of the year. If the DT parameter is omitted, default is the current date.
TM=hhmm	Time to modify the A, C, or M option, in a format indicating hours and minutes on a 24-hour clock. If the TM parameter is omitted, default is 0000 which is midnight preceding the day specified by the DT parameter.
PN=pkid-list	List of 1 through 16 pack identifiers, separated by commas, of packs on which files are to be reloaded. If the PN parameter is omitted, default is all active packs.
LO=x	Indicator of type of audit information to be written: F — Full information. P — Partial information. Default.
OU=lfn/len/dc	File to which reload status information is to be written. lfn — Name of file. Must be 1 through 8 letters or digits beginning with a letter. Default is OUTPUT. len — Number of small pages in file. When /len is omitted, default is #40. dc — Disposition code indicating processing of file: PR — Print on any available printer at the end of the utility When /dc is omitted, the utility does not cause the file to be printed.

Figure 4-17. LOADPF Control Statement Format (Sheet 2 of 2)

specified, a file must meet all criteria specified before it is loaded. Similarly, the options specified by the OP parameter operate in combination to select files. A parameter OP=CM, for example, selects files created or modified. The UN and POOL parameters interact as shown in table 4-2.

A dump tape or file produced by a nonprivileged DUMPF contains a copy of the formatted PFI prior to opening each file being dumped; therefore, the access fields are not updated on the dump tape/file but are updated on the file index maintained by the system. An attempt to reload the files by date and time of last access uses the original values for those fields for the comparison.

A dump tape or file produced by a privileged DUMPF contains a copy of the formatted PFI after opening each file being dumped; therefore, the access fields are updated on both the dump tape/file and the file index maintained by the system. An attempt to reload the files by date and time of last access uses the updated values for those fields for the comparison.

LOADPF can execute concurrently with other tasks, including other LOADPF tasks. It produces, as output, a list file containing the names of files loaded, and any appropriate error messages, including the file name of the file being processed. Attached or unattached status is not reported on the output file.

NORERUN (SET NORERUN STATUS)

The NORERUN control statement is valid only within a batch deck. It is executed directly by the batch processor. It changes the default status for the file created by the STORE card from rerun to norerun, such that when the system is brought up after a system failure the batch deck is destroyed. In the absence of this control statement, all jobs in a batch job set are rerun from the beginning when the system is brought up after a system failure that terminated the batch processor.

Rerun or norerun status affects an entire batch deck and not simply a single job within that deck.

NORERUN format is shown in figure 4-18. An example of NORERUN use is shown in figure 4-19.

```
NORERUN.
```

Figure 4-18. NORERUN Control Statement Format

OLE (OBJECT LIBRARY EDITOR)

The OLE control statement operates with modules produced by assembly or compilation of source programs. It can be used to create:

Library file. The library contains a directory of module names and entry points, as well as the modules, and can be used by the loader to satisfy externals. When the library is referenced by the LIBRARY parameter of LOAD, the loader can selectively load modules from the library to satisfy externals or the ENTRY parameter of LOAD.

Modmerge file. This file is a collection of modules without a directory. When the file is referenced in a LOAD control statement, all modules are loaded, subject to the loading order specified in the LOAD control statement. A modmerge file offers the programmer convenience in referencing a group of modules and also overcomes the restriction of a limit of 10 files that can be referenced in a LOAD control statement.

OLE can also be used to list modules and characteristics of the modules on library or modmerge files independent of any file creation.

OLE produces a single file in either library or modmerge file format. As many as 50 input files can be specified. Input files must contain a library, one or more modules output by a CYBER 200 assembler or compiler, or a modmerge format file of several modules. Input files must be attached, and output files are local.

OLE format is shown in figure 4-20. Parameters can appear in any order, but subparameters cannot be separated.

During OLE execution, modules are placed in the new file in the order they are encountered in the input files as listed with the INPUT parameter. Any modules specified by the OMIT parameter are not made a part of the new file. If duplicate module names exist in the input files, only the first module encountered is written to the new file.

POOL FILE UTILITIES

The seven pool file utilities create, access, and destroy a pool of files that can be accessed by other users. Section 3 explains how a pool is created and used.

```
STORE...
EGJOB.
.
.
.
COMMENT.  JOB RERUNS IF A FAILURE OCCURS HERE
NORERUN.
.
.
.
COMMENT.  JOB DOES NOT RERUN IF A FAILURE OCCURS HERE
RERUN.
.
.
.
COMMENT.  JOB RERUNS IF A FAILURE OCCURS HERE
.
.
.
6/7/8/9 card
```

Figure 4-19. NORERUN/RERUN Example

OLE, INPUT=lfn-list, {NEWLIB=liblfn / MODMERGE=modlfn}, OMIT=sfn,mod-list, LIST=opt, OUTPUT=lfn/len.

INPUT=lfn-list	List of 1 through 50 file names, separated by commas, whose modules are to be written to the file specified by the NEWLIB or MODMERGE parameter. If the INPUT parameter is omitted, only the parameters LIST and OUTPUT are valid.
NEWLIB=liblfn	Name of file to contain new library being created. Must be 1 through 8 letters or digits beginning with a letter and can duplicate a file name specified with the INPUT parameter. If the NEWLIB and MODMERGE parameters are omitted and INPUT is specified, default is NEWLIB.
MODMERGE=modlfn	Name of file to contain modmerge file being created. Must be 1 through 8 letters or digits beginning with a letter. Can duplicate the name of a file specified with the INPUT parameter. No default name exists.
OMIT=sfn,mod-list	Indicator that listed modules (mod-list) of file sfn are to be omitted from the library or modmerge file. Multiple OMIT parameters can appear in a single OLE parameter list.
LIST=opt	Indicator of files to be listed by module names, length, creation date, and entry point names: lfn-list — List of file names, separated by commas, whose contents are to be listed. The destination file is listed without being specified. 0 — Suppress all listings. If the LIST parameter is omitted, only the library or modmerge file is listed. Output appears on the file specified by the OUTPUT parameter.
OUTPUT=lfn/len	File to which listing is to be written. lfn — Name of file. Must be 1 through 8 letters or digits beginning with a letter. When the OUTPUT parameter is omitted, default is OUTPUT. len — Number of small pages in file. When /len is omitted, default is #10.

Figure 4-20. OLE Control Statement Format

All these utilities send error and status messages either to the dayfile of a batch job or to the terminal of an interactive user.

Pool utilities follow in alphabetical order.

PACCESS POOL UTILITY

The PACCESS utility can only be executed by the pool boss. It establishes the list of users who are authorized to access all files in the pool. Format is shown in figure 4-21.

PATTACH POOL UTILITY

The PATTACH utility attaches an authorized user to a pool. It is required before any file in the pool can be accessed. Format is shown in figure 4-22.

PACCESS, poolname, USER=userno.

poolname	Name of existing pool.
USER=userno	Identification of users who are to have pool access: ALL — Universal access to all users u-list — List of 1 through 32 user numbers, separated by commas, who can access pool.

Figure 4-21. PACCESS Control Statement Format

PATTACH, poolname.

poolname	Name of existing pool to which user is to be attached.

Figure 4-22. PATTACH Control Statement Format

One user can be attached to as many as four pools simultaneously. Depending on the parameters used by the pool boss when a file was given to the pool, access to any given pool file might be limited to reading only.

PCREATE POOL UTILITY

The PCREATE utility establishes a pool. The user defining a pool name becomes the pool boss and has responsibility for entering files in the pool, specifying who can use the pool, and destroying the pool. Format is shown in figure 4-23.

PCREATE, poolname.

poolname	Name of pool to be established. Must be 1 through 8 letters or digits beginning with a letter.

Figure 4-23. PCREATE Control Statement Format

PDELETE POOL UTILITY

The PDELETE utility removes users from the list of authorized users. It can only be executed by the pool boss. Format is shown in figure 4-24.

PDELETE, poolname, USER=userno.

poolname	Name of existing pool.	
USER=userno	Identification of users whose access permission is to be rescinded:	
	ALL	Universal access removed
	u-list	List of 1 through 16 user numbers, separated by commas, to be removed. Invalid when universal access is set.

Figure 4-24. PDELETE Control Statement Format

A user cannot be removed from the list of authorized users while attached to the pool. Individual users cannot have their access to a pool deleted if universal access has been granted.

PDESTROY POOL UTILITY

The PDESTROY utility destroys a pool. It can only be executed by the pool boss, who has the responsibility of first removing all files from the pool. Format is shown in figure 4-25.

PDESTROY, poolname.

poolname	Name of existing pool to be destroyed.

Figure 4-25. PDESTROY Control Statement Format

The pool cannot be destroyed while users or the pool boss are attached or any files exist in the pool. Any pool file that is to be preserved as a permanent file must be copied to a private file before it is removed from the pool.

PDETACH POOL UTILITY

The PDETACH utility detaches a user from a pool. It should be executed when files in a pool are no longer required. Format is shown in figure 4-26.

PDETACH, poolname.

poolname	Name of pool from which user is to be detached.

Figure 4-26. PDETACH Control Statement Format

PFILES POOL UTILITY

The PFILES utility produces a list of information about pools in the system. Specific information listed depends on the parameters selected. Format is shown in figure 4-27.

PFILES, {poolname / USER=userno}.

poolname	Name of existing pool for which authorized users are to be listed.	
USER=userno	Identification of pool bosses:	
	ALL	All pool names are to be listed along with the pool boss and the number of users currently attached to each pool.
	u-list	List of 1 through 16 pool boss user numbers, separated by commas. All pools belonging to each pool boss are to be listed by name with the number of users currently attached to the pool.

Figure 4-27. PFILES Control Statement Format

If information being reported exceeds the size of the display screen of an interactive terminal, the last line of the display instructs the terminal user to enter MORE or YES to continue the display.

Figure 4-28 shows output.

```
A.  PFILES(USER=ALL)

    POOL NAME      POOL BOSS      USER COUNT

    DEMO           300299         0
    TESTPOOL       333322         2
    STESTS         333333         0

PFILES UTILITY COMPLETED

B.  PFILES(DEMO)

USERS GRANTED ACCESS TO POOL

    300045   300047   300034   300089
    000022
```

Figure 4-28. PFILES Sample Output

PURGE (EVICT PERMANENT OR POOL FILES)

The PURGE control statement releases mass storage space assigned to one or more permanent files. It can also be used by a pool boss to destroy pool files. The utility deletes appropriate entries from the pack file index for packs on which files reside and releases space for reassignment. PURGE control statement format is shown in figure 4-29.

PURGE,lfn-list,CL=POOL,ST=xxx,optional CYBER front-end parameters.

lfn-list	List of 1 through 16 CYBER 200 permanent file names, separated by commas, to be purged, or the name of one CYBER front-end file to be purged. CYBER 200 file names must be 1 through 8 letters or digits, beginning with a letter (except for drop files). lfn-list must appear before other parameters. For pool files, list of 1 through 16 CYBER 200 file names, separated by commas, of files named in attached pools. The CL=POOL parameter is also required. Files can be members of different pools. Private and pool files cannot be mixed in a single list.
CL=POOL	Optional parameter that indicates that all files named in lfn-list are CYBER 200 pool files. The user must be the pool boss and be attached to all pertinent pools.
ST=xxx	Optional parameter that indicates that the file in lfn-list is a CYBER front-end file resident on the mainframe designated by the ST parameter.
optional CYBER front-end parameters	(The ST parameter is not used when CYBER 200 files are to be purged.)

Figure 4-29. PURGE Control Statement Format

If the ST parameter is specified, PURGE is effective for a CYBER file on the designated mainframe.

If a permanent file lfn, specified by PURGE, is currently attached to another suffix, the file is demoted from permanent to local. It then exists as a local file under that suffix and is governed by all rules pertaining to local files.

For a successful PURGE of pool files, the user must be the pool boss for the first attached pool containing the pool file lfn.

Files cannot be purged while they are open to a task. If the system cannot purge a file, it returns error messages to the dayfile of a batch job or to the terminal of an interactive user.

READCC (READ ALTERNATE CONTROL CARD FILE)

The READCC control statement is valid only in a batch job. It is executed directly by the batch processor. It causes the batch processor to read control statements from an attached file other than the one containing the READCC control statement.

READCC control statement format is shown in figure 4-30.

READCC, lfn.

lfn	Name of file containing control statements.

Figure 4-30. READCC Control Statement Format

The specified file must contain ASCII data in unstructured format and contain only control statements. Any valid control statement can appear, including images of other READCC statements. READCC can be nested at as many as eight levels. Reading from the specified file terminates when an ASCII end-of-file (#1C) is encountered within the data. Control then returns to the statement following the appropriate READCC.

REQUEST (CREATE LOCAL FILE)

The REQUEST control statement defines a local mass storage file. REQUEST can be used to ensure that a file is created on a particular pack. It can also be used to determine whether a given pack has adequate space to hold a file of the required size; if not, the utility returns a fatal error code. This utility controls whether mass storage allocated to the file is contiguous at creation and whether the file can be extended. REQUEST cannot create a file if a local or attached permanent file with the same name exists.

Execution of REQUEST results in allocation of mass storage.

The NOSEGMENT and NOEXTEND parameters are used to control the continuity of the file. The interaction between the NOSEGMENT and NOEXTEND parameters is as follows:

Extendable File	Segmentable File	Result
No	No	One segment. File cannot be extended.
No	Yes	File created as one or two segments. File cannot be extended.
Yes	No	File created as one segment. Noncontiguous segments can be added.
Yes	Yes	File created as one or two segments. Noncontiguous segments can be added.

REQUEST control statement format is shown in figure 4-31. The first parameter must be the file name. File length, if specified, must be the second parameter. All other parameters are optional and can appear in any order.

No message is returned for successful completion.

Retention period for the file is an installation option. The SWITCH control statement can be used to specify a particular number of days the file is to be retained on mass storage.

RERUN (SET RERUN STATUS)

The RERUN control statement is valid only within a batch deck. It is executed directly by the batch processor. It reverses the effects of a preceding NORERUN control statement that appears in any job in the batch job set. When status is rerun, all jobs in a batch job set are rerun from the beginning (at whatever priority they were running) when the system is brought up after system failure that terminated the batch processor.

REQUEST, lfn/len, ACCESS=acs, TYPE=typ, SECURITY=lvl, PACK=packid, NOEXTEND, NOSEGMENT.

lfn	Name of mass storage file to be created. Must be 1 through 8 letters or digits beginning with a letter.
/len	Number of small pages to be allocated for the file stated as a decimal or hexadecimal number 1 through #FFFF. If this parameter is omitted, default is 8.
ACCESS=acs	File access permission: R Read W Write R and W can be combined in any order without commas. If the ACCESS parameter is omitted, default RW.
TYPE=typ	File type: C Virtual code file P Physical data file. Default.
SECURITY=lvl	Security level of file being created. Must be in the range of 1 through 255. If the SECURITY parameter is omitted, default is the level resulting from STORE or LOGON use.
PACK=packid	Identifier for pack on which file is to be created. Pack identifiers are six characters in length: fewer characters are blank padded on the right, while a character string longer than six characters is truncated. If the PACK parameter is omitted, default is a pack selected by the system.
NOEXTEND	Indicator that the file cannot be extended. That is, the file size established by the length parameter is an absolute size. If the NOEXTEND parameter is omitted, the file can be increased to an additional percentage of original file length, as determined by an installation parameter.
NOSEGMENT	Indicator that file space on mass storage must be contiguous when the file is created. If the NOSEGMENT parameter is omitted, the system might allocate initial file space in two segments.

Figure 4-31. REQUEST Control Statement Format

Rerun and norerun status affects all jobs in a batch deck and not simply a single batch job within the deck. That is, any RERUN or NORERUN control statement affects the processing of all batch jobs between a STORE card and a subsequent 6/7/8/9 card. At the beginning of the first batch job in the set, job status is rerun status; at the beginning of a subsequent job in the set, however, status is that in effect at the end of the preceding job in the same set.

RERUN format is shown in figure 4-32. Figure 4-19 shows RERUN use.

```
RERUN.
```

Figure 4-32. RERUN Control Statement Format

An installation parameter can override the rerun capabilities such that no jobs are rerun.

RETURN (EVICT LOCAL FILES OR DETACH PERMANENT FILES)

The RETURN control statement releases mass storage space assigned to one or more local files. It can also be used to disassociate an attached permanent file from a suffix. RETURN control statement format is shown in figure 4-33.

RETURN, { lfn-list / * } .

lfn-list — For private files, list of 1 through 16 names separated by commas, of files to be destroyed or detached. Each file name must be 1 through 8 letters or digits, beginning with a letter (except for drop files).

* — Indicator that all local and attached permanent files belonging to the user suffix are to be destroyed and detached, respectively.

Figure 4-33. RETURN Control Statement Format

Files cannot be returned while they are open to a task. The file must exist at this user suffix before it can be returned. If the system cannot evict a local file or disassociate an attached file from a suffix, it returns an error message to the dayfile of a batch job or to the terminal of an interactive user. When the utility is called with the * parameter, all files are returned with the exception of any batch input file open to the batch processor. The job itself might terminate abnormally, however, when required files are not available.

ROUTE (SPECIFY FILE DISPOSITION)

The ROUTE utility controls file disposition. Only local and attached permanent files can be routed. It is required when a print or punch file is to be directed to a specific device.

ROUTE control statement format is shown in figure 4-34. The first parameter must be the logical file name; all other parameters are optional and can appear in any order.

If the file is to be routed to an Access Station, only the first seven characters of the file are sent to the station as the file name.

SWITCH (CHANGE FILE CHARACTERISTICS)

The SWITCH control statement changes any of the following characteristics of a local or an attached permanent file: file name, type, access, and retention days. For files that can be executed, this utility can also be used to indicate the length of the drop file that is to be created when the file is called for execution. The drop file length is placed in the executable file's minus page and in other system tables. A privileged user can change the drop file length of a public file.

SWITCH control statement format is shown in figure 4-35. The first parameter is required; newlfn, if specified, must appear as the second parameter. All other parameters are optional and can appear in any order.

TCOPY (TAPE COPY AND FILE REFORMAT)

The TCOPY control statement copies information between magnetic tape and mass storage, between two mass storage files, or between two magnetic tape files. This utility is required when copying to or from magnetic tape, although the DUMPF utility can be used to dump a file to tape with no change in format. During a copy with TCOPY, data format can be converted in any of the combinations marked with X in table 4-3. Input files must be attached, and output files are local.

TABLE 4-3. TCOPY CONVERSION POSSIBILITIES

From	To			
	Binary	BCD	ASCII	Blocked BCD
Binary	X	X	X	X
BCD	X	X	-	-
ASCII	X	-	X	-
BLOCKBCD	X	-	-	-

Reformatting performed by the utility during the copy operation is controlled by the programmer through directives. The directives allow the copy operation to be controlled on a file or record basis and provide for manipulation of the file being copied. More than one input or output file can be manipulated by a single set of directives, so that a given file can be separated or merged into others.

Directives have these functions:

OPEN	Describe input and output files
COPYR, COPYF	Copy by records or files
BKSPR, BKSPF	Backspace records or files

ROUTE, lfn, DC=dc, { SAVE / DEF }, IC=ic, FID=fid, EC=ec, CM=cm, ST=st, TID=tid, OT=ot, DI=di, optional CYBER 200 Link parameters.

lfn — Logical file name of file to be routed.

DC=dc — File disposition: Punch in format indicated by IC and EC parameters.

- SC — Scratch; that is, the file is to be destroyed. Default.
- PR — Print in format indicated by IC parameter.
- PU — For files routed to an Access Station, punch in format indicated by EC parameter. For files routed to a Unit Record Station, punch in format of default disposition.
- IN — For files routed to an Access Station or CYBER front-end, place file in the input queue. Cannot be used to route file to CYBER 200 input queue.
- P1 — Print on 501 printer.
- P2 — Print on 512 printer.
- LR — Print on 580-12 printer.
- LS — Print on 580-16 printer.
- LT — Print on 580-20 printer.

  (P1, P2, LR, LS, LT: Valid for CYBER 200 Link only.)
- PF — For files routed to an Access Station, save as permanent file.

DEF — Indicator that the file is not to be disposed of by this ROUTE call. (All other parameters of the call are processed and values retained in internal system tables.) The file will subsequently be disposed of by any of the following occurrences:

A ROUTE call that omits the DEF parameter

Program termination

If the DEF parameter is omitted, the file is released to the appropriate queue at the time the utility executes and the job can no longer reference the file.

SAVE — Indicator that a copy of the file is to be made and routed. The copy of the file is a local file with the name Q5ROUTE. SAVE is the default for permanent files.

IC=ic — Internal characteristics of the file:

- AS — Unstructured file with ASCII data; and if DC=PR, the file has ANSI carriage control characters.
- PA — Unstructured file with ASCII data; and if DC=PR, the file has ASCII carriage control characters. Default.
- BI — Unstructured file with binary data. Required for files to output as binary punch files.

FID=fid — First five characters of the file name while the file is in the output queue; must be 1 through 5 letters or digits beginning with a letter. The system adds two sequence characters as the sixth and seventh characters of the file name. The eighth character for files to be output as a Unit Record Station is blank. If the file is SAVEd, default is the file name.

EC=ec — Punch or print file external characteristics:

Punch:
- 26 — 026 keypunch format
- 29 — 029 keypunch format; default
- 80 — 80-column binary format
- *B — CYBER 200 binary format

(*B can only be used on files routed to the Unit Record Station)

Print:
- B4 — Print file on BCD 48-character print train.
- B6 — Print file on BCD 64-character print train.
- A4 — Print file on ASCII 48-character print train.
- A6 — Print file on ASCII 64-character print train.
- A9 — Print file on ASCII 95-character print train.

CM=cm — For files to be routed to an Access Station or CYBER 200 Link Station, conversion mode:

- DI — CYBER front-end display 6-bit code (64-character set). Default.
- EC — CYBER 6-bit extended display code (128-character set). (Access Station only.)
- BI — Binary. No conversion.

ST=st — Mainframe identifier of the system where the file is to be output: This is an installation-defined identifier. AST is reserved for the Access Station.

TID=tid — Identifier of terminal to which file is to be returned. The identifier for the CYBER 200 Link consists of two letters or digits. TID=0, TID, or TID=any other unrecognizable identifier causes file to be routed to the central site. Not meaningful for the Unit Record Station. The identifier for the Access Station consists of 1 to 7 letters and digits.

OT=ot — For files to be routed to an Access Station, origin type for terminal:

- B — Local batch. Default.
- E — Remote batch.

DI=di — Optional CYBER 200 Link parameters. See CYBER 200 Link Reference Manual. For files routed to an Access Station, 1 to 8 alphanumeric characters.

Figure 4-34. ROUTE Control Statement Format

SWITCH, oldlfn, newlfn, TYPE=typ, ACCESS=acs, RETENTION=days, DROP=dlen.

oldlfn	Name by which existing local or attached permanent file is known. Required.
newlfn	Name to which oldlfn is to be changed. Must be 1 through 8 letters or digits beginning with a letter and must not duplicate the name of an existing local or permanent file (attached or unattached).
TYPE=typ	New file type: C Virtual code file P Physical data file
ACCESS=acs	New file access permission: R Read permission granted W Write permission granted R and W can be combined in any order.
RETENTION=days	Number of days file is to be retained on mass storage. Must be a decimal integer 0 through 1023.
DROP=dlen	Number of small pages in drop file to be created when a virtual code file is executed. Must be a decimal or hexadecimal integer 0 through #FFFF.

Figure 4-35. SWITCH Control Statement Format

SKIPR, SKIPF	Skip forward records or files
WEOF	Write end-of-file on output file
CLOSE	Close file
REWIND	Rewind files
END	Terminate TCOPY

When a binary output file is being produced, disk files can be in either SRM-structured format or unstructured format. File content is treated as binary data without regard to contents. When a coded output tape is being produced from a mass storage file, however, the file must be in unstructured format and contain ASCII data with ASCII control characters.

Tape files described with an OPEN directive parameter of ASCII, BIN, or BCD equate a physical record between interblock gaps with a logical unit corresponding to that terminated with the ASCII unit separator #1F. Files described by a BLOCKBCD parameter of OPEN are files compatible with S format tapes produced by the NOS or NOS/BE operating systems. They correspond to tape structure described by CYBER Record Manager E type blocks with F type records in which whole records are packed into a block defined by a maximum block size.

A tape file described with an OPEN directive parameter of BLOCKBCD can be copied to another tape by describing the output tape as BIN. TCOPY has no tape label capabilities. A directive can skip any label. TCOPY control statement format is shown in figure 4-36.

TCOPY, lfn.

lfn	Name of file containing directives for TCOPY. Default is INPUT.

Figure 4-36. TCOPY Control Statement Format

The directive file referenced must have unstructured format and contain ASCII data. An ASCII end-of-file marker (#1C) terminates the file. Maximum file size is nine small pages.

DIRECTIVES

Directives execute in the order they are entered interactively through a terminal or they appear in the directive file. The first reference to any file must be an OPEN directive. All directives reference a file by the name specified in OPEN. The last directive must be END, which terminates TCOPY.

OPEN Directive

The OPEN directive must be the first reference to a file. If the file is an existing mass storage file, it is opened; or if the file does not exist, it is created. The utility sends appropriate messages to the operator of a station with tape units to have the proper tape made available. OPEN format depends on whether a tape file or a mass storage file is referenced.

For a mass storage file, OPEN format is:

OPEN,lfn,DISK,len,packid

lfn	Name of an existing mass storage file or a mass storage file to be created.
DISK	Indicator that the file is a mass storage file. DISC is an alternative spelling.
len	Hexadecimal length of a new file to be created, in number of small pages. Default is #200.
packid	Identifier of the disk pack on which a new file is to be created. If omitted, the system selects an available pack.

For a tape file, OPEN format is as follows. Parameters must be in the order shown. Default parameters are selected by successive commas.

OPEN,lfn,TAPE,tracks,mode,density,blocsiz,flen,ac,vsn

lfn	Name to be used to reference a tape file. Must be 1 through 8 letters and digits beginning with a letter.

TAPE Indicator that the file is a tape file.

tracks Type of tape unit:

7	7-track tape
9	9-track tape

mode Tape format:

ASCII	Coded 7-track or 9-track
BCD	Coded 7-track
BIN	Binary 7-track or 9-track. Default.
BLOCKBCD	Blocked-record format.

density Recording density:

2	200 bpi 7-track
5	556 bpi 7-track
8	800 bpi 7-track or 9-track
16	1600 bpi 9-track

Default is the density at which the unit is set.

blocsiz Hexadecimal number of characters per physical tape record:

For ASCII or BCD mode, maximum number of characters in the variable-size physical records. Maximum is #7FF0. Default.

For binary mode, number of characters in each record. Maximum is #7FF0.

For BLOCKBCD mode, number of characters in a record. Default is #1400, which is equivalent to 5120 decimal characters.

flen For BLOCKBCD mode only, hexadecimal number of characters in each logical record. Maximum is #3FF. Default is #88.

acs Write ring:

RI	Write ring in for output file
RO	Write ring out for read-only input file. Default

vsn Volume serial number of tape volume. Must be 1 through 6 characters.

As many as 16 files can be open at one time when TCOPY executes interactively; as many as 14 files can be open at one time when TCOPY executes from a batch job. More than 16 or 14 files could be manipulated in a single TCOPY execution, however, since the CLOSE directive can close files no longer needed.

Other Directives

Directives are presented below in alphabetical order. The default value for n in all cases is n=1.

BKSPF,lfn,n

The BKSPF directive backspaces a file. For a tape file, n tape marks are backspaced over, leaving the tape positioned immediately before the last tape mark. For a disk file, the position indicator is set back n file separators defined by #1C.

BKSPR,lfn,n

The BKSPR directive backspaces n records. For a tape file, n physical records are backspaced over. For a disk file, the position indicator is set back n unit separators defined by #1F.

CLOSE,lfn,RUN

The CLOSE directive closes a file. The RUN parameter is optional; it causes a tape file to be rewound and unloaded.

COPYF,inlfn,outlfn

The COPYF directive copies all data from file inlfn to file outlfn. For a tape, a file is defined as all information between two tape marks. For a disk being copied to a coded tape, a file is defined as all information between two file separators defined by #1C. Otherwise, a disk file is defined as all information in the pages allocated to the file.

COPYR,inlfn,outlfn,n

The COPYR directive copies one or more records from file inlfn to file outlfn. Copying begins at the current position of each specified file. For a tape file, a record is defined as all information between two interblock gaps; that is, a physical record.

REWIND,lfn

The REWIND directive rewinds the named file. A tape file is rewound to its load point. A disk file is repositioned to the beginning of mass storage allocated to the file.

SKIPF,lfn,n

The SKIPF directive positions forward by files. For a tape file, n tape marks are skipped, leaving the tape positioned after the tape mark. For a disk file, the position indicator is set forward n file separators defined by #1C.

SKIPR,lfn,n

The SKIPR directive positions forward by records. For a tape file, n physical records are skipped forward. For a disk file, the position indicator is set forward n unit separators defined by #1F.

WEOF,lfn

The WEOF directive writes an end-of-file indicator. For a tape file, a tape mark is written. For a disk file, a file separator #1C is written.

DETAILED COPY OPERATIONS

Table 4-4 summarizes TCOPY operations. They are described in detail below.

TABLE 4-4. SUMMARY OF TCOPY PROCESSING

Copy Operation	General Description	Padding	Tape Mark FS Written After		Blank Compression/ Expansion
			COPYF	COPYR	
Disk to Disk	No scanning of data.	No	Yes	Yes	No
Disk to Binary Tape	No scanning of data.	Yes	Yes	Yes	No
Disk to BCD or ASCII Tape	US becomes tape block.	No	Yes	If TM found yes	Yes
Disk to Blocked BCD	US blocked fixed size.	No	Yes	Yes	Yes
Binary Tape to Disk	Same as disk to binary tape. Concatenate on disk only if multiple copy operations to same disk file.	Yes	Yes	Padding produces FS	No
BCD/ASCII Tape to Disk	Tape block defines US.	Yes	Yes	Yes	Yes
Blocked BCD to disk	Fixed size record becomes US.	Yes	Yes	Yes	Yes
Binary Tape to Binary Tape	No scanning of data.	Yes	Yes	Yes	No
Binary Tape to BCD/ASCII Tape	US becomes tape block.	No	Yes	If TM found yes	Yes
BCD/ASCII Tape to Binary Tape	Tape block defines US Concatenated to block size.	Yes	If TM found yes	If TM found yes	Yes
ASCII Tape to ASCII Tape BCD Tape to BCD Tape	Copy without translation or conversion.		If TM found yes	If TM found yes	No

Disk to Disk

The contents of a disk file are not scanned during a copy to another mass storage file. Copying stops when the end of the shorter file has been reached.

Disk to Binary Tape

The contents of a disk file are not scanned during a copy to a binary tape. Rather, the information in the disk file is written to the tape in fixed-size physical records. The blocsiz parameter of the OPEN directive establishes the physical record size. The last physical record on the tape is padded, if necessary, with a bit pattern corresponding to #1C, so that all physical records are the same size.

For COPYR, copying stops after n physical records have been written. For COPYF, copying stops at the end of mass storage allocated to the file. Two tape marks are written on the tape file after COPYF. TCOPY backspaces over one of the terminating tape marks.

Disk to BCD or ASCII Tape

The disk file to be copied must be unstructured format with ASCII data. Each unit of the data becomes an individual physical record on tape. The ASCII control character #1F, which is the unit separator in the data, is not written to the tape.

Data in the disk file must be at least four characters in length. Any unit separator that indicates a physical record smaller than four characters, or larger than the tape block size maximum, terminates the copy operation with an error.

If the data in the disk file has blank compression, the blanks are expanded before being written to the tape. Blank compression is indicated by the ASCII control character #1B followed by a character formed by the number of compressed blanks plus #30.

A COPYR operation normally terminates when n physical records have been written. The presence of a file separator, #1C, in the disk file causes a tape mark to be written to the tape file and the copy operation to terminate.

Binary Tape to Disk

Copy of a binary tape concatenates the physical records of the tape onto the disk file without regard to contents. Any tape mark on the input file is not copied to mass storage.

A COPYF operation terminates when all space allocated to the disk file has been written or a tape mark is encountered in the input file.

A COPYR operation terminates when n physical records have been copied to the disk file. It also terminates if a tape mark is encountered before n physical records are copied.

For both copy operations, the last page of the disk file is padded, if necessary, with a bit pattern corresponding to #1C; the pattern is overlaid, however, if more data is copied to the file.

BCD or ASCII Tape to Disk

Copy of an ASCII or BCD tape creates an unstructured disk file with ASCII data. The unit separator #1F is appended to each physical record of the tape and concatenated onto the disk file. When two or more blanks are encountered within a record on the tape file, they are compressed as described above before being written to the disk.

The last page of the disk file is padded, if necessary, with a bit pattern corresponding to #1C; the pattern is overlaid, however, if more data is copied to the file.

Binary Tape to Binary Tape

Copy of a binary tape to a binary tape occurs without regard to data contents. The physical record size on the output file can be different from that of the input file. Otherwise, a given number of characters from the input file are blocked into physical records on the output file according to the blocsiz parameter of the OPEN directive for the output file. The last tape record is padded, if necessary, with a bit pattern corresponding to #1C.

A COPYR operation terminates when n physical records have been read from the input file. A tape mark on the input file terminates either a file or record copy.

Binary Tape to BCD or ASCII Tape

The binary tape to be copied must contain ASCII data complete with ASCII control characters. Each unit of the data, as indicated by the unit separator #1F, becomes an individual physical record on tape. The ASCII control character #1F is not written to the output tape, however.

A COPYR operation normally terminates when n physical records have been written. The presence of a file separator, #1C, in the input data causes a tape mark to be written to the output file and the copy operation to terminate.

BCD or ASCII Tape to Binary Tape

Copy of an ASCII or BCD tape creates a binary file with ASCII control characters. Each physical record from the input tape is assumed to be a unit, and the unit separator #1F is appended to the data before it is written to the binary tape. Binary tapes must have fixed physical record size, as established by the blocsiz parameter of the OPEN directive. Each physical record written is the same size without respect to whether a unit separator coincides with the physical record end. When two or more blanks are encountered within a record on the input file, they are compressed as described above before being written to the binary file.

A COPYR operation normally terminates when n physical records have been written. The presence of a tape mark on the input file during either a record or file copy causes a tape mark to be written to the binary file and the copy operation to terminate with an error. The last record of the file is padded, if necessary, with a bit pattern corresponding to #1C so that all physical records are the same size.

ASCII Tape to ASCII Tape/BCD Tape to BCD Tape

These copy operations occur without any translation or conversion.

Disk to Blocked BCD Tape

The disk file to be copied must be unstructured format with ASCII data. Each ASCII unit indicated by #1F is expanded as necessary to the fixed record size indicated by the flen parameter of the OPEN directive. Expansion occurs through restoration of blank compression or through addition of blanks at the end of the record. The fixed-length logical records are then blocked according to the blocsiz parameter of the OPEN directive and written to the tape as one physical record. The unit separator #1F is not part of the written data. Any unit on the disk file that is longer than fixed record size causes the copy to terminate. The last block in the file is not padded.

Blocked BCD Tape to Disk

Copy of a blocked BCD tape creates an unstructured file with ASCII data. Each physical record on the tape contains a number of fixed-length logical records. The copy operation compresses blanks from each logical record, appends the unit separator #1F, and writes the record to the disk file.

TV (TERMINATION VALUE CHECK)

The TV control statement is valid only within a batch job. It resets a termination value that controls whether or not the job continues normally when an error code is returned at job step completion.

Each job step returns a code to the batch processor at the completion of that step. The batch processor compares the returned code with the current termination value:

If the returned value is less than or equal to the current termination value, the job continues normally.

If the returned value is greater than the current termination value, abnormal job termination procedures occur. The batch processor either terminates the job or resumes job execution at the control statement following an EXIT control statement. The termination value is set to 255 when an EXIT control statement established established the execution path.

If the system aborts a task, the TV value is not checked, however. In this instance the job terminates or resumes after any EXIT control statement.

The TV control statement can also be used to test a particular termination value against the highest return code from any previous job step. In this instance the programmer test is independent of the test made by the batch processor at the end of each job step. The test for the highest return code initiates abnormal job termination procedures when the highest return code exceeds the termination value.

TV control statement format is shown in figure 4-37.

The initial termination value for the job is established by the TV parameter of the job statement. The highest return code value is 0 at the beginning of the job.

Utilities described in this manual return only codes 0, 4, or 8, although higher values might be returned in extraordinary circumstances:

- 0 Successful completion
- 4 Nonfatal errors encountered
- 8 Fatal errors encountered

The value that a given task returns is established by a TERMINATE message executed within that task.

TV, value+.

value	Termination value 0 through 255.
+	Indicator of a value set, as distinguished from a value to be tested immediately (optional).

Figure 4-37. TV Control Statement Format

UPDATE is a utility for maintaining and manipulating a mass storage file containing images of coded punch cards or their equivalent. The utility provides the programmer with features that are a subset of the UPDATE capabilities available under the NOS or NOS/BE operating systems. The UPDATE card image file cannot be interchanged between these systems, however, since internal file structures differ between the operating systems.

Typical use of UPDATE involves maintenance of a group of FORTRAN subroutines or assembly language routines. For convenience, the programmer often specifies each routine as a separate deck, so that one routine can be changed or extracted without affecting other routines in the file. Because each card image in the deck has its own UPDATE-supplied sequence number, it can be referenced individually. A card can be deleted and replaced by two others, for example, in order to correct a routine or to increase its functions. At programmer request a deck can be extracted from the card image file in a format acceptable to a compiler or assembler and used as if it had been entered into the system as a punch deck. Once a source card is in the UPDATE card image file, any physical punch card can be dispensed with.

A source deck that is to be maintained through UPDATE must be made a part of a special format file known as a program library. Creation of a program library is accomplished through UPDATE itself. Subsequently, the program library can be changed on an UPDATE correction run: new decks can be added, existing decks removed, or the contents of any deck changed.

The contents of a deck need only be images of coded cards. UPDATE makes no assumptions about card contents. While programs are customary contents, they are not required contents and UPDATE is equally applicable to a set of data cards or any other text.

The programmer controls UPDATE operations through the parameters on the UPDATE control statement and through a file containing directives and text. The directives are supplementary instructions for UPDATE; the text is source cards to be made part of the UPDATE card image file. Together, the directives and text are called the input stream.

EXAMPLES

An example of an UPDATE creation run in which several FORTRAN routines become a program library with three decks is shown in figure 5-1. The UPDATE control statement indicates a new library is to be created with the name MYDECKS. The input for UPDATE is specified as the file INPUT, which is also the default file name when the I parameter is omitted.

The first directive encountered in figure 5-1 is *DECK: therefore, UPDATE recognizes a creation run and begins construction of a new program library. All cards following *DECK, up until the second *DECK directive, are written as a deck with the name MAIN. The first card is assigned the identifier MAIN.2, the next MAIN.3, and so forth. (The *DECK directive itself is also part of the library and has the identifier MAIN.1.)

```
STORE card
job statement
UPDATE,I=INPUT,N=MYDECKS.
DEFINE,MYDECKS.
7/8/9
*DECK MAIN
     PROGRAM MAIN . . .
     :
     END
*DECK SUBPROG
     SUBROUTINE SUB1 . . .
     :
     SUBROUTINE SUB2 . . .
     :
*DECK ERRPROG
     SUBROUTINE SUB3
     :
6/7/8/9
```

Figure 5-1. Typical UPDATE Creation Run

A new deck, with card identifiers in the form SUBPROG.n, begins when UPDATE encounters the second *DECK directive. In this example, the main program is in a deck with the same name as the program, two subroutines are in a deck with the name SUBPROG, and a third subroutine is in a deck with the name ERRPROG. At the end of the UPDATE run, a program library exists with three decks.

The DEFINE control statement makes the local file MYDECKS a permanent file. The file MYDECKS remains in the system after job termination.

Figure 5-2 shows a correction run using the program library created in figure 5-1. This example adds a new card with text DIMENSION UP(50) near the beginning of subroutine SUB2. Notice that the location of the insertion is identified by the card identifier assigned within the deck SUBPROG and not by the subroutine name.

```
STORE card
job statement
UPDATE (Q,P=MYDECKS)
FORTRAN (I=COMPILE)
LOAD.
GO.
7/8/9
*IDENT FIXIT
*INSERT SUBPROG.89
        DIMENSION UP(50)
*COMPILE SUBPROG,MAIN
6/7/8/9
```

Figure 5-2. Typical UPDATE Correction Run

The UPDATE control statement in figure 5-2 identifies the existing program library with the P parameter; it also instructs UPDATE to operate in quick mode rather than full mode since the Q parameter appears. When UPDATE begins execution, it reads the next unexecuted record of the batch job, which is presumed to contain the input stream. The first card in this stream gives a name (FIXIT) to the corrections being made. The second card identifies the location at which the new card is to be added; namely, after the card with the identifier SUBPROG.89. Since the third card in the input stream does not correspond to a directive, it is considered a text card. Within the program library it becomes identified as FIXIT.1. The last card in the input stream instructs UPDATE to write decks MAIN and SUBPROG to a file in a format suitable for input to the FORTRAN compiler. By default, in the absence of a C parameter on the UPDATE control statement, the file name is COMPILE.

The FORTRAN compiler call in figure 5-2 names a file COMPILE as having the program to compile. Output of compilation is then loaded and executed with the LOAD and GO control statements.

Neither the FORTRAN call nor execution is required in figure 5-2. They are shown only to provide the programmer with a source listing of active cards in the deck with their card identifiers or to confirm proper program execution.

GENERAL PROCESSING

During execution, UPDATE manipulates several files known as the input file, new program library, source file, old program library, compile file, and the listable output file. File operations depend on whether UPDATE is performing a creation run or a correction run.

An UPDATE run is defined as all operations with a program library that results from a single UPDATE call. Any given run is a creation run or a correction run:

A creation run constructs a program library. It is the original transfer of punch cards or card images into UPDATE format.

A correction run changes an existing program library. As a result of the run, a new program library might be generated; but the program library is new only in the sense that the changes are incorporated into the existing program library. All history information remains.

File names are specified by parameters of the UPDATE control statement, which are summarized in table 5-1. Table 5-2 shows a summary of UPDATE directives used during a run.

UPDATE MODE AND FILES

All files used by UPDATE must reside on mass storage; all files created by UPDATE reside on mass storage. Any of these files can be stored on magnetic tape, but the programmer is responsible for any transfers between tapes and mass storage devices. Default length of all of these files is #100 small pages. Any of the UPDATE default files are opened and used if they exist as attached permanent files. If the files do not already exist, UPDATE uses Q7GETFIL to create them as local files.

TABLE 5-1. SUMMARY OF UPDATE CALL PARAMETERS

Parameter	Function
C	Specify name of compile file.
D	Define compile file card image width excluding UPDATE sequence information.
F	Select full update mode and source file and compile file contents.
I	Specify name of file with input stream.
L	Select listable output file contents.
N	Specify name of new program library file.
O	Specify name of listable output file; content is determined by L parameter.
P	Specify name of old program library file.
Q	Select quick update mode.
S	Specify name of source file; content includes common decks and is determined by mode.
T	Same as S, but omits common decks.
8	Define compile file card image width including UPDATE sequence information.
*	Redefine master control character for directives.
/	Redefine control character for comments.

An intermediate processing file created by UPDATE when a new program library is being created has the file name TEMNEWPL. Its default length is #400 small pages or the length of the new program library, whichever is larger.

The files that UPDATE creates or uses are described below. An input file is always required; an old program library file is also required for a correction run. Each of these files has a default file name, but any other name can be specified through the appropriate parameter on the UPDATE control statement.

The content of any compile file, source file, or new program library produced during a correction run is affected by the UPDATE mode. The mode of an UPDATE run is determined by a combination of the omission or specification of the F and Q parameters on the UPDATE control statement.

TABLE 5-2. SUMMARY OF UPDATE DIRECTIVES

Directive Keyword Abbreviation	Directive Format	Use
*AF	*ADDFILE lfn,ident.seqnum	Read creation directives and text from named file and insert after card identified.
*CA	*CALL deck	Write common deck to compile file.
*CD	*COMDECK deck,NOPROP	Define common deck and propagation parameter.
*C	*COMPILE deck1,deck2,...,deckn	Write specified decks to compile file, source file, and new program library.
	*COMPILE deck1.deck2	Write inclusive range of decks to these files.
*DK	*DECK deck	Define deck to be included in program library.
*D	*DELETE ident1.seqnum,ident2.seqnum	Deactivate inclusive range of cards.
	*DELETE ident.seqnum	Deactivate specified card.
*ID	*IDENT idname,B=n,K=ident,U=ident	Define correction set, bias for seqnum, and whether specified correction sets must be known or unknown to process this set.
*I	*INSERT ident.seqnum	Write subsequent text cards after card identified.
*PD	*PURDECK deck1,deck2,..., deckn	Permanently remove specified decks from program library.
	*PURDECK deck1.deck2	Permanently remove inclusive range of decks.
*P	*PURGE idname1,idname2,...,idname3	Permanently remove specified correction sets from program library.
	*PURGE idname1.idname2	Permanently remove inclusive range of correction sets.
	PURGE idname,	Permanently remove specified correction set and all sets introduced after it.
*RD	*READ lfn	Read directives and text from specified file.
*Y	*YANK idname1,idname2,...,idnamen	Temporarily remove specified correction sets from program library.
	*YANK idname1.idname2	Temporarily remove inclusive range of correction sets.
*YD	*YANKDECK deck1,deck2,...,deckn	Temporarily deactive decks specified.
*/	*/ comment	Copy text to listable output file.

Normal (selective), full, or quick UPDATE mode is selected by:

Parameter	Mode
Both F and Q omitted	Normal selective mode in which the only decks processed are those modified or otherwise selected for processing.
F specified	Full mode in which all decks on the old program library are processed.
Q specified	Quick mode in which only decks specified on COMPILE directives are processed.
Both F and Q specified	Quick mode.

Input File

The input file contains the input stream. The input stream consists of directives that provide the details of UPDATE processing and any new cards to be added to the program library. File name is specified by the I parameter of the UPDATE call; default file name is INPUT.

New Program Library

The new program library is the file of card images and internal information in a special format that can be processed only by UPDATE. It contains a deck list of the names of all decks in the file and a directory of all correction sets introduced into the file. Each card is represented in a format that adds a card identifier and adds history and status information known as correction history bytes. Blanks are compressed out of the card image.

A new program library is an output file created by UPDATE. Initially, it is generated on a creation run. For subsequent correction runs, the previous new program library is used as an input file and identified as the old program library; a new program library that incorporates the changes made during a correction run is then output from the correction run. File name is specified by the N parameter of the UPDATE call; default file name is NEWPL.

Source File

The source file is a file output during a correction run. It consists of card images that would allow regeneration of a new program library in resequenced format during a subsequent creation run. Only active cards and decks are part of the source file. File name is specified by the S parameter of the UPDATE call; default file name is SOURCE. The content of the file is controlled by the T, F, and Q parameters. The programmer is responsible for routing the file to a punch or other output device.

Old Program Library

The old program library is the file generated as a new program library in a previous creation or correction run. It contains a record of changes made since the program library was created. It is required for any correction run. File name is specified by the P parameter of the UPDATE call; default file name is OLDPL. The old program library is an unstructured file with blank compression.

Compile File

The compile file is an output file that contains a copy of a deck in the program library restored to a format that can be processed by a compiler or assembler. Only active cards in the deck are part of the compile file. File name is specified by the C parameter of the UPDATE call; default file name is COMPILE. The content of the file is controlled by the directives and the F or Q parameters of the UDPATE call, with the D or 8 parameter selecting the number of columns in the image of each card. The compile file is an unstructured file with blank compression.

List File

The listable output file is the print file containing information for use by the programmer. It shows the card identifiers assigned by UPDATE which the programmer must use to reference a card image in any future correction run. File name is specified by the O parameter of the UPDATE call; default file name is OUTPUT. Content of the file is controlled by the L parameter, with options that can select a listing of directives processed, errors, comments, and a list of card images in the program library.

CREATION OF PROGRAM LIBRARY

A creation run exists when the first directive of the input file, other than a comment, is DECK or COMDECK. If the first directive is READ and the first directive of the file being read is DECK or COMDECK, a creation run also exists. Even if an old program library file is assigned to the job, UPDATE ignores its existence and processes the run in creation mode.

Directives that can be used in a creation run are limited to:

READ

DECK

COMDECK

ADDFILE

In a creation run, each DECK or COMDECK directive defines a deck to be inserted into the program library under construction. UPDATE decks can be one of two types, regular decks or common decks. They differ in that common decks can be called by name so that they are inserted into the text of another deck when the compile file is being generated; one copy of the common deck exists on storage, but multiple copies can be part of an output file.

In practice, the text written to a program library is often FORTRAN or assembly language routines in their punch card format. UPDATE considers all cards to be a string of characters and takes no recognition of card contents. For convenience, a programmer often assigns a different UPDATE deck name to each routine but there is no requirement to do so. UPDATE divides text cards into decks following directive instructions.

The order of decks in the program library is controlled by the programmer; decks appear in the order in which they are found in the input stream. A common deck must precede any regular deck that might call the common deck.

All cards following a DECK or COMDECK directive, up until the next DECK or COMDECK directive, are considered to be part of the deck and each receives a unique sequence number. The directive defining the deck itself is assigned a sequence number 1. Any READ directive among the text cards causes UPDATE to temporarily stop reading from the current input stream and to read from the specified file until an end-of-file is encountered; reading then resumes from the main input stream. Text cards read from the file specified by READ are numbered as if they were part of the original input text.

CARD IDENTIFICATION

The image of each card stored in a deck contains information known as correction history bytes. This information, which is generated by UPDATE, maintains a history and status of a card and is the means by which UPDATE can reverse status. Deletion of a card, for example, is accomplished by the addition of a correction history byte to the card image rather than a physical deletion of the image. Consequently, the card can be reactivated at some later time. Only purge operations are irreversible.

A DECK or COMDECK directive is written to the program library as part of the deck text. Consequently, these directives can be referenced just as any other card in the text. Deactivating a DECK directive, for example, has the effect of making its following text a part of the deck that precedes it in the library.

UPDATE recognizes one full form and two short forms of card identifiers. The full form card identifier is:

ident.seqnum

ident.	1 through 8 character name of a correction set or deck. A period terminates the ident name.
seqnum	Decimal ordinal (1 through 65 535) representing the sequence number of the card within the correction set or deck. Any character other than 0 through 9 terminates the sequence number.

The two short forms of card identifiers can be used on INSERT or DELETE directives. The short forms are expanded as follows:

seqnum	Expands to idname.seqnum where idname is a correction set identifier, whether or not it is also a deck name.
.seqnum	Expands to dname.seqnum where dname is a deck name.

In the short form, idname is assumed to be the last explicitly named ident given on an INSERT or DELETE directive, whether or not it is a deck name. The dname is assumed to be the last explicitly named ident given on an INSERT or DELETE directive that is known to be a deck name. Both of these default idents are originally set to YANK$$$ so the first directive using a card identifier must use the full form to reset the default.

All deck names are also idents (but all idents are not decks). Thus, if EXAMPLE is the deck name last used, and there is no subsequent explicit reference to a correction set identifier, then both .281 and 281 expand to EXAMPLE.281 as the identifier. If there is an explicit reference to a correction set identifier after the explicit reference to the deck name, then 281 would expand to the correction set identifier, while .281 would expand to EXAMPLE.281 as the identifier.

Figure 5-3 shows differences in identifier expansion depending on the order of directive records, assuming A is a deck name and B is a correction set identifier on an UPDATE old program library.

```
*ID C
*INSERT A.2
     data card
*INSERT B.1
     data card
*D  2,  3        expands to *DELETE B.2, B.3
*D  4,  .5       expands to *DELETE B.4, A.5
*D  .7, 5        expands to *DELETE A.7, B.5
*D  .9, .10      expands to *DELETE A.9, A.10

whereas:

*ID D
*INSERT B.1
     data card
*INSERT A.2
     data card
*D  2,  3        expands to *DELETE A.2, A.3
*D  4,  .5       expands to *DELETE A.4, A.5
*D  .7, 5        expands to *DELETE A.7, A.5
*D  .9, .10      expands to *DELETE A.9, A.10
```

Figure 5-3. Card Identifier Expansion

CORRECTION RUN

A correction run, which is the most common use of UPDATE, introduces changes into the existing program library. UPDATE recognizes a correction run, as opposed to a creation run, under either of the following circumstances:

The first directive, other than a comment, is IDENT.

The first directive, other than a comment, is READ or ADDFILE and the first directive on the alternate file is IDENT (in the case of READ) or DECK or COMDECK (in the case of ADDFILE).

All directives can be used during a correction run.

The IDENT directive establishes a name for the correction set. Any cards inserted into the library are sequenced within this name. On subsequent correction runs, individual cards in the correction set can be referenced by sequence number. The entire correction set can also be referenced as a whole.

When a new program library is being generated, all corrections must be part of a correction set with the exception of PURGE, PURDECK, and ADDFILE. That is, IDENT must be the first directive other than a comment. If READ is the first directive, the alternate input file must have IDENT as its first directive. If a new program library is not being generated (that is, routines are being extracted, but no changes made), directives can appear without a correction set identifier.

Four directives need not be part of a correction set. They are directives PURGE, PURDECK, and ADDFILE which cause the current set to be terminated, and COMPILE. The COMPILE directive, as with the comment directive, can appear anywhere within or outside a correction set. It is not processed until all corrections have been made.

More than one correction set can be introduced during a single run. The correction set established by the first IDENT directive remains in effect until UPDATE either encounters another IDENT directive or encounters a PURGE, PURDECK, or ADDFILE directive. The subsequent IDENT directive establishes a second correction set name.

A correction run can include the addition of new decks to the program library when a new program library is created. Decks to be added are identified by a DECK or COMDECK directive following an INSERT, DELETE, or ADDFILE directive.

Deck List and Directory Order

UPDATE maintains a list of all decks in the program library which is known as a deck list. The order of entries in the deck list is under programmer control; original deck list entries correspond to the order in which decks are

written during the creation run. Subsequent additions of decks are made at the location specified by the programmer with a preceding INSERT, DELETE, or ADDFILE directive; or if the directive preceding DECK or COMDECK is IDENT, at the end of the current library.

The location of an entry in the deck list is significant in terms of parameters for PURDECK, YANKDECK, and COMPILE directives in which a range of decks is referenced. The order of names in a range reference must be the same as the order in the deck list. The decks named and all those between are then processed in accordance with the directive. An error exists if they are in reverse order.

Similarly, as each correction set is introduced into the program library, UPDATE creates an entry in an internal directory in chronological sequence. The location of an entry in the directory is significant in terms of parameters for PURGE and YANK directives in which a range of correction sets is referenced. The order of reference must be the same as the order of the directory. The identified correction sets and all those between are processed in accordance with the directive. An error exists when a correction set range is not referenced in the order the sets were introduced into the library.

PURGE and YANK Directives

The two purge directives are PURGE and PURDECK: the first operates on all cards identified by correction set name while the second operates on all cards within an identified deck. (Introduction of a new deck on a creation run must be made as part of a correction set, but that addition usually is the only change within that correction set.) The two yank directives are YANK and YANKDECK. As with the purge directives, the former operates with correction sets and the latter with decks.

The purge directives differ from the yank directives in that yank operations are temporary. Cards yanked from the program library are temporarily deactivated. They can be reactivated by a subsequent yank of the yank directive that inactivated the cards.

In contrast, any change made to a program library through a purge directive is permanent. A reversal of a purge operation is possible only through the reintroduction of the cards into the library as if they had not previously existed.

Since the YANK directive itself must be introduced as part of a correction set, a future correction set that references the correction set containing the YANK reactivates the original correction set. For instance:

To inactivate all cards added by IDENT PSR003:

```
*IDENT TAKEOUT
*YANK PSR003
```

To reactivate the same cards:

```
*IDENT PUTBACK
*YANK TAKEOUT
```

Overlapping correction messages might be produced as a result of these procedures.

UPDATE stores all YANK directives in a deck with identifier YANK$$$. Individual cards in the deck, but not the entire deck, can be referenced. An alternative to the two directives above that yanked identifier TAKEOUT, is:

```
*PURGE YANK$$$.TAKEOUT
```

Once a card or an entire correction set has been yanked, it cannot again be referenced except for a reactivation request. That is, a yanked card cannot be referenced on DELETE or INSERT.

Overlapping Corrections

UPDATE can detect four overlapping correction situations. When any of these types is detected, UPDATE prints the offending line with the words TP.n OVLP appended on the far right.

Type	Meaning
Type 1	Two or more modifications are made to one card by a single correction set.
Type 2	A modification attempts to activate an already active card.
Type 3	A modification attempts to deactivate an already inactive card.
Type 4	A card is inserted after a card which was inactive on the OLDPL.

Detection of an overlap does not necessarily indicate a user error. Overlap messages are advisory, and they point to conditions in which the probability of error is greater than normal.

Type TP.2 and TP.3 are detected by comparing existing correction history bytes with those to be added. Complex operations involving YANK and PURGE might generate these overlap messages even though no overlap occurs.

Modifications for each correction set are performed by UPDATE in the order in which sets are introduced. The order is irrelevant if no correction is dependent on another. If a dependent relationship exists, however, order is of paramount importance.

UPDATE DIRECTIVES

Directives are instructions for UPDATE to follow in creating its output files. A directive must begin in column 1 with the master control character. Each directive has both a full keyword and an abbreviated keyword as shown in table 5-2.

General format is:

```
*keyword p-list
```

* Master control character which distinguishes a directive from a text card. Must appear in column 1. This character can be changed through the *=c parameter of the UPDATE control statement.

keyword	Name of one of the UPDATE directives or an abbreviation for a directive. No blanks can occur between the master control character and the keyword; a comma or blank terminates the keyword.
p-list	Parameters identifying decks, cards, or files. Multiple blanks can appear between the keyword and parameters. Parameters in the list are separated by commas; embedded blanks cannot appear in the list. Notice that several parameters contain a period as part of a single parameter.

No terminator appears at the end of a directive.

The master control character is recorded in the program library. For a correction run the master control character should match the character used when the program library was created; if the characters do not match, UPDATE uses the character stored as part of the program library.

Any card in the input stream that cannot be recognized as a directive or as a comment is assumed to be text.

Directives are described below in alphabetical order.

ADDFILE DIRECTIVE

The ADDFILE directive causes UPDATE to add a file of new decks to the new program library. It differs from the READ directive in that contents of the specified file is limited to those that add decks. The first card of the specified file must be a DECK or COMDECK directive. No directives other than comments, DECK, or COMDECK can appear in the file.

ADDFILE directive format is shown in figure 5-4. If only one parameter appears, it is assumed to be lfn.

*ADDFILE lfn,ident.seqnum

lfn	Name of file from which decks are to be added. Default is the file specified by the I parameter of the UPDATE call.
ident.seqnum	Identifier of card after which decks are to be placed on the program library. If omitted, the addition is made at the end of the program library.

Figure 5-4. ADDFILE Directive Format

When the specified file is not INPUT, UPDATE reads directives and text cards until an end-of-file (#1C) is encountered. UPDATE then returns to the file specified by the I parameter of the UPDATE call and continues processing the main input stream. When the file specified on the ADDFILE directive is INPUT, however, UPDATE reads directive and text cards only until either an end-of-file or an UPDATE directive other than DECK, COMDECK, or CALL is encountered.

An ADDFILE directive cannot appear among directives read from a file specified by a READ directive. Otherwise, it can appear anywhere in the input stream, but its appearance terminates the current correction set.

CALL DIRECTIVE

The CALL directive causes UPDATE to write the text of a previously encountered common deck onto the compile file. The directive itself is stored as part of a deck and can be referenced by its sequence number. It is effective only within a deck.

CALL directive format is shown in figure 5-5.

*CALL deck

deck	Name of an existing common deck to be written to the compile file.

Figure 5-5. CALL Directive Format

Neither the CALL directive nor the COMDECK directive which defined the deck becomes part of the compile file.

Common decks can call other common decks, but a common deck must not call itself or a deck that contains a call to the common deck.

COMDECK DIRECTIVE

The COMDECK directive establishes a common deck that can be called from other decks as they are being written to the compile file. It is one of the two directives that establishes the existence of a creation run. The directive can be used in any correction run to add a common deck to a particular location in the program library.

COMDECK directive format is shown in figure 5-6.

*COMDECK deck,NOPROP

deck	Name of common deck being added. Must be 1 through 8 characters A through Z, 0 through 9, or + - / * ( ) $ = ___. Must not duplicate the name of an existing deck.
NOPROP	Indicator that decks calling this common deck are not to be considered as modified when the common deck itself is modified; that is, the effects of common deck changes are not be propagated during a normal UPDATE mode. Optional.

Figure 5-6. COMDECK Directive Format

The COMDECK directive itself is part of the program library and has a sequence number of 1 within the name established by the directive. For a creation run, the deck order in the input stream determines the location of the common deck in the program library. For a correction run, the location in the program library is determined by the preceding INSERT directive or the location resulting from a preceding DELETE or ADDFILE. Common decks need not appear first on the program library, but they must appear before any decks from which they are called during a creation run.

The NOPROP parameter of the COMDECK directive that created a common deck determines whether a deck calling a corrected common deck is to be considered as having been corrected also.

COMPILE DIRECTIVE

The COMPILE directive affects the decks to be written to the compile file and any new program library or source file during normal or quick UPDATE mode. The directive is ignored during a full UPDATE.

Normal mode	Decks specified on COMPILE directives and corrected decks are written to the compile file
Quick mode	Decks specified on COMPILE directives and any common decks they call are written to the compile file

The directive also affects the contents of any new program library and source file, as shown in table 5-3.

COMPILE directive format depends on whether decks to be written are specified individually by name or are specified as a range of deck names as shown in figure 5-7.

Decks always are written in the order that the decks exist on the old program library.

COMPILE directives can appear anywhere within the input stream. They do not affect the current correction set name.

DECK DIRECTIVE

The DECK directive establishes a deck in the program library. It is one of the two directives that establishes the existence of a creation run. The directive also can be used in any correction run to add a deck to the location indicated by a preceding ADDFILE directive.

A. Compile listed decks

*COMPILE deck1,deck2, . . . , deckn

deck	Name of deck to be written to the compile file, new program library file, and source file.

B. Compile range of decks

*COMPILE deck1.deck2

deck1.deck2	Names of first and last decks in range, inclusive, to be written to the compile file. The name of deck1 must appear in the old program library deck list before deck2.

Figure 5-7. COMPILE Directive Format

DECK directive format is shown in figure 5-8.

Each deck must have a unique name within the program library.

The DECK directive itself is part of the program library and has a sequence number of 1 within the name established by the directive.

*DECK deck

deck	Name of deck. Must be 1 through 8 characters A through Z, 0 through 9, or + - / * ( ) $ = __. Must not duplicate the name of any other deck in program library.

Figure 5-8. DECK Directive Format

TABLE 5-3. FILE CONTENTS AND UPDATE MODE

File	Normal Mode Contents	Full Mode (F) Contents	Quick Mode (Q) Contents
New program library	All regular and common decks after corrections made in sequence of old program library.	Same as normal mode source file.	Decks specified on COMPILE directives, any common decks they call, and any common decks encountered on old program library prior to all decks of COMPILE.
Compile File	All decks corrected or listed on COMPILE directives, and any deck calling a corrected common deck (unless NOPROP specified on COMDECK).	All active decks on old program library.	All decks on COMPILE directives and any common decks they call.
Source File	All currently active DECK, COMDECK, and CALL directives and active text required to recreate library.	Same as normal mode source file.	Currently active cards required to create new program library resulting from quick mode.
T parameter excludes common decks.			

DELETE DIRECTIVE

The DELETE directive deactivates a card or group of cards and optionally adds text cards following the directive. A deactivated card remains on the library and retains its sequencing. It can be referenced just as if it were not deactivated. A deactivated card is not written to any compile file or source file, however.

DELETE directive format depends on whether cards to be deactivated are specified by card identifier or by a range of cards, as shown in figure 5-9.

A. Delete specified card

*DELETE ident.seqnum

ident.seqnum	Card identifier for single card to be deleted.

B. Delete range of cards

*DELETE ident1.seqnum,ident2.seqnum

ident1.seqnum, ident2.seqnum	Card identifiers of first and last cards, inclusive, in sequence of cards to be deleted. Card ident1.seqnum must appear before ident2.seqnum in the existing library. The range can include cards in a deactivated state.

Figure 5-9. DELETE Directive Format

IDENT DIRECTIVE

The IDENT directive establishes the name for the set of corrections being made. Cards added in this correction set are sequenced within the name specified. Any card whose status is changed by this set receives a correction history byte that references the name from IDENT. All correction set names must be unique.

IDENT directive format is as shown in figure 5-10.

*IDENT ident,B=num,K=ident,U=ident

ident	Name to be assigned to this correction set. Must be 1 through 8 characters A through Z, 0 through 9, or + - / * ( ) $ = ___. Must not duplicate the name of another correction set or deck.
B=num	Bias to be added to sequence numbers within deck.
K=ident	Indicator that specified correction set name must exist in the directory of the library before corrections can be made.
U=ident	Indicator that specified correction set name must not exist in the directory of the library.

Figure 5-10. IDENT Directive Format

The B, K, and U parameters can appear in any order. More than one K or U parameter can be specified; in this instance, all correction set names specified must meet the criteria before the correction set is processed. If the criteria of these parameters is not met, UPDATE skips the correction set and resumes processing with the next IDENT, PURGE, PURDECK, or ADDFILE directive.

INSERT DIRECTIVE

The INSERT directive adds text cards following it to the program library at the location specified.

INSERT directive format is as shown in figure 5-11.

New cards receive card identifiers established by the correction set name of the preceding IDENT directive.

*INSERT ident.seqnum

ident.seqnum	Card identifier of card after which the insertion is to be made.

Figure 5-11. INSERT Directive Format

PURDECK DIRECTIVE

The PURDECK directive permanently removes a deck or group of decks from the program library. Every card in the deck is purged, regardless of what correction set it might belong to. Purging, unlike yanking, cannot be rescinded.

PURDECK directive format depends on whether decks to be purged are specified individually by deck name or by a range of deck names, as shown in figure 5-12.

A. Purge decks listed

*PURDECK deck1,deck2, . . . , deckn

deck	Name of deck to be purged. Names can appear in any order.

B. Purge range of decks

*PURDECK deck1.deck2

deck1.deck2	Names of first and last decks, inclusive, to be purged. Names must appear in the relative order in which decks exist in the deck list.

Figure 5-12. PURDECK Directive Format

A PURDECK directive can appear anywhere in the input stream, but its appearance terminates the current correction set. Any directive following PURDECK must be another purge directive or a directive that institutes another correction set. The deck YANK$$$ cannot be purged.

The name of the purged deck can be reused as a deck name. It can be used as a new correction set identifier only if it does not already exist in the directory.

PURGE DIRECTIVE

The PURGE directive permanently removes a correction set or group of correction sets from the program library. Every card in the correction set is purged, regardless of its status as active or inactive. Purging, unlike yanking, cannot be rescinded.

PURGE directive format depends on whether correction sets to be purged are specified individually by correction set name, by a range of correction set names, or by relative time of introduction into the program library, as shown in figure 5-13.

A PURGE directive can appear anywhere in the input stream, but it terminates the current correction set. Any directive following PURGE must begin a new correction set.

A. Purge listed correction sets

*PURGE ident1,ident2, . . . ,identn

ident	Identifier of a correction set to be purged. Identifiers can appear in any order.

B. Purge range of correction sets

*PURGE ident1.ident2

ident1.ident2	Identifiers of first and last correction sets, inclusive, to be purged. Identifiers must appear in the relative order in which the correction sets were introduced into the program library; that is, they must appear in the order they exist in the directory.

C. Purge later correction sets

PURGE ident,

ident	Identifier of correction set to be purged along with all correction sets introduced after the specified correction set.
*	Indicator that the program library is to return to an earlier level. Intervening purge directives prevent complete return.

Figure 5-13. PURGE Directive Format

READ DIRECTIVE

The READ directive causes UPDATE to temporarily stop reading the current input stream and to begin reading an input stream from the file specified on the READ directive. READ differs from ADDFILE in that the content of the file specified by READ is not restricted except to prohibit the appearance of either another READ directive or an ADDFILE directive. UPDATE reads from the specified file until an end-of-file (#1C) is encountered. Processing then continues with the main input stream.

READ directive format is as shown in figure 5-14.

*READ lfn

lfn	Name of alternate file containing input stream.

Figure 5-14. READ Directive Format

YANK DIRECTIVE

The YANK directive temporarily removes a correction set or group of correction sets from the program library. Cards activated by the correction set are deactivated; cards deactivated by the correction set are reactivated. YANK differs from purge in several respects: YANK must be part of a correction set; it does not terminate the current correction set; its effects can be rescinded.

YANK directive format depends on whether correction sets to be yanked are specified individually by correction set name or by a range of correction set names, as shown in figure 5-15.

UPDATE places the YANK directive in the YANK$$$ deck. If a correction has been yanked, it is ignored during compile file or source file generation.

A. Yank listed correction sets

*YANK ident1,ident2, . . . ,identn

ident	Identifier of a correction set to be yanked. Identifiers can appear in any order.

B. Yank range of correction sets

*YANK ident1.ident2

ident1.ident2	Identifiers of first and last correction sets, inclusive, to be yanked. Identifiers must appear in the relative order in which the correction sets were introduced into the program library; that is, they must appear in the order they exist in the directory.

Figure 5-15. YANK Directive Format

YANKDECK DIRECTIVE

The YANKDECK directive temporarily deactivates all cards within the decks specified. All cards are deactivated, regardless of the correction set to which they belong. YANKDECK differs from PURDECK in several respects: YANKDECK must be part of a correction set; it does not terminate the current correction set; its effects can be rescinded.

YANKDECK directive format is as shown in figure 5-16.

*YANKDECK deck1,deck2, . . . ,deckn

deck	Name of deck to be yanked. Names can appear in any order.

Figure 5-16. YANKDECK Directive Format

The deck YANK$$$ cannot be deactivated as a whole. Individual YANK directives within this deck can be yanked by a YANK directive, however.

/ COMMENT DIRECTIVE

The / directive introduces a comment into the listable output file. UPDATE ignores this card except to copy it to the output file. A comment can appear at any place in the input stream.

/ comment directive format is shown in figure 5-17. The slash must appear in column 2. Column 3 must be a comma or blank.

```
*/ comment
```

Figure 5-17. / Comment Directive Format

The slash can be redefined as another character through the /=c parameter of the UPDATE call.

UPDATE CONTROL STATEMENT

The format of the control statement that calls UPDATE to execution is shown in figure 5-18. All parameters are optional and can appear in any order. A comma must separate parameters.

```
UPDATE, {C | C=file}, D, F, {I | I=lfn}, L=opt, {N | N=lfn/#nnn}, {O | O=lfn/#nnn}, {P | P=lfn}, Q, {S | S=lfn/#nnn},
{T | T=lfn/#nnn}, 8, *=c, /=c.
```

C — Compile file name. The content of the compile file is determined by the UPDATE mode.

omitted or C	Decks are written to the file named COMPILE.
C=lfn or C=lfn/#nnn	Decks are written to file named lfn. File length is #100 small pages or the number of pages specified by nnn.
C=PUNCH	Decks are written to file named PUNCH. The D and 8 parameters are implied.
C=0	Compile file suppressed.

D — Data width on compile file excluding UPDATE sequence identifiers.

omitted	72 columns of data.
D	80 columns of data.

F — Full UPDATE mode.

omitted	Normal selective UPDATE mode, as long as Q is not specified. The compile file contains only those decks corrected in this run or otherwise specified on COMPILE directives.
F	Full UPDATE mode. The compile file contains all active decks in the program library.

I — Input stream file name.

omitted or I	Directives and text are on the file named INPUT.
I=lfn	Directives and text are on file named lfn.

L — Listable output options to be written to file named with the O parameter.

omitted	For a creation run, options A, 1, and 2. For a correction run, options A, 1, 2, 3, and 4.
L=c. . . c	Each character in string c . . . c selects one of the following options.
A	Error decks, correction set identifiers, common decks, and decks written to the compile file are listed.
F	Full listing.
0	The character 0 overrides any other options specified and suppresses the entire listing.

Figure 5-18. UPDATE Control Statement Format (Sheet 1 of 3)

1 Suppress deck name list, identifier list, and continuous commentary when L=1 is specified.

List errors if this option is selected by omission of the L parameter.

2 List directives with ****** preceding each directive with valid format. Each IDENT directive begins a new page of the listing.

3 Comment on each card changed. Comments include the deck name, card image, card identifier and sequence number, and an indicator of action taken for that card:

I Card added.
A Inactive card reactivated.
D Active card deactivated.
P Card purged. If the card was active, ACTIVE also appears.

4 List cards of input stream established by directives. Cards read as a result of a READ directive are identified to the right with the file name; cards inserted as a result of an ADDFILE directives are listed only when option 4 is explicitly selected. *ERROR* accompanies any cards in error.

9 List all active and inactive cards with status:

I Inactive.
A Active.

Option 3 overrides option 9.

L=0 Suppress all listings.

N New program library file name.

omitted In a correction run, suppress new program library generation. In a creation run, write a newpl on the file named NEWPL.

N Write new program library on file named NEWPL.

N=lfn or N=lfn/#nnn Write new program library on file named lfn. File length is #100 small pages or the number of pages specified by nnn.

O Listable output file name.

omitted or O Write list output to file named OUTPUT.

O=lfn or O=lfn/#nnn Write list output to file named lfn. File length is #100 small pages or the number of pages specified by nnn.

P Old program library file name.

The P parameter is valid only for a correction run.

omitted or P Old program library resides on file named OLDPL.

P=lfn Old program library resides on file named lfn.

Q Quick UPDATE mode. The source file and the new program library are described in table 5-3.

omitted When F is also omitted, normal selective UPDATE mode.

Q Only those decks specified on COMPILE directives are processed. Corrections to decks not specified on COMPILE are ignored, except for ADDFILE. The compile file contains only decks referenced on COMPILE directives and the common decks they call.

The Q parameter takes precedence when both F and Q are specified.

Figure 5-18. UPDATE Control Statement Format (Sheet 2 of 3)

S	Source file name. The content of this file is determined by the UPDATE mode.	
	omitted	Suppress source output file unless it is selected by the T parameter.
	S	Source output file to be written on file named SOURCE.
	S=lfn or S=lfn/#nnn	Source output file to be written on file named lfn. File length is #100 small pages or the number of pages specified by nnn.
T	Omit common decks from source file. The content of the source file is determined by the UPDATE mode, with the T parameter excluding common decks.	
	omitted	Suppress source output unless it is selected by the S parameter.
	T	Source output file to be written on file named SOURCE, with common decks excluded.
	T=lfn or T=lfn/#nnn	Source output file to be written on file named lfn, with common decks excluded. File length is #100 small pages or the number of pages specified by nnn.
	The T parameter takes precedence over the S parameter.	
8	Card image width on compile file including UPDATE sequence identifiers	
	omitted	90-column card image, which preserves columns 73 through 80 of original card
	8	80-column card image, with UPDATE sequence information in columns 73 through 80
*	Master control character for directives	
	omitted	* is the first character of each directive
	*=c	c is the first character of each directive for this UPDATE run. c can be any character A through Z, 0 through 9, or + - * / $ or =. If the character specified for a correction run is not the same as the character used when the old program library was created, the old program library character is used.
/	Comment control character in column 2	
	omitted	Comment control character is /
	/=c	c is the comment control character. c can be any character A through Z, 0 through 9, or + - * / $ or =.

Figure 5-18. UPDATE Control Statement Format (Sheet 3 of 3)

The CYBER 200 operating system includes several sets of functions that can be called from a user program. These capabilities are:

Checkpoint/restart. This call allows a program to be restarted from a point well into execution if the program terminates abnormally.

Magnetic tape subroutines. These routines allow a program to process standard tape labels and otherwise improve tape access.

System Record Manager. These functions give a programmer many of the capabilities of assembly language system messages for file access. The functions can be accessed by assembly language programs also.

CHECKPOINT/RESTART

The checkpoint/restart capability allows a task to be restarted from some intermediate point in the event of abnormal task termination. The facility provides a means of conserving machine time from a second execution of a long task. Through a call to CHKPNT, a programmer captures the status of an executing task and any of its controllees; if necessary, the chain of tasks can be restarted from the checkpoint information captured by the call.

When checkpoint is called, the system makes copies of the drop files associated with the checkpointed task and all of its controllees. Checkpoint information includes a list of controllees to be restarted. Checkpoint file names are formed by the system modifying the file name in the checkpoint call with an ending digit 1 through 5 indicating the controllee level relative to the task with the checkpoint call.

Checkpoint restores system message control for the checkpointing task and tasks at lower levels in its controllee chain. Any message outstanding when a checkpoint occurs is not preserved, however. For a controllee with a message bypass pointing to a controller of higher level than the checkpointing task, the bypass is set by restart to point to the checkpointing task.

See section 4, Message Control, of the CYBER 200 Operating System Reference Manual, Volume 2 of 2.

CHKPNT (CHECKPOINT CALL)

A checkpoint is taken by the system in response to a call to CHKPNT. The programmer is responsible for ensuring that the checkpoint occurs at a logical breaking point during task execution.

CHKPNT subroutine format is shown in figure 6-1.

All files created by CHKPNT are local files. The local files can be made permanent with DEFINE.

CALL CHKPNT (lfn, rst, err, erlfn)

lfn — Name of file to which the checkpoint information is to be written. Must be 1 through 8 letters or digits beginning with a letter expressed as a Hollerith constant or a variable left-justified and blank filled.

rst — Variable to which the system returns a response after checkpoint and restart:

- 0 Checkpoint executed
- 1 Restart executed

err — Variable to which the system returns a code for any error found in either checkpoint or restart

erlfn — Variable to which the system returns the name, left-justified and blank filled, of any file in which errors were encountered during checkpoint or restart. The particular error that causes this variable to be set is documented in the err variable.

Figure 6-1. CHKPNT Subroutine Format

A task can call CHKPNT more than once with a given file name for the lfn variable. At each such call to CHKPNT, a copy of the current state of the drop file is created. When CHKPNT terminates successfully, this copy is assigned the name lfn, and the previous checkpoint file is destroyed. If program execution is aborted before checkpoint terminates, the previously existing file is retained as the checkpoint file.

The intermediate checkpoint file name at the calling controllee level is Q5CKP1; if lower level controllees exist, their intermediate checkpoint file names are Q5CKP2 through Q5CKP5.

Checkpoint error responses that can be returned in the err variable established by the call to CHKPNT are:

- 0 No error
- 1 Checkpoint file name specified in erlfn duplicates an existing local or attached permanent file name
- 2 Input/output connector error
- 3 Mass storage or system table space not available
- 4 Invalid file name supplied by program
- >4 System error with return word formatted as in STATUS call of SRM
- -1 Error in bound implicit map entry for file OUTPUT

Restart error responses that can be returned in the err variable established by the call to CHKPNT are:

0	No error
10	Controllee cannot be initialized because system tables are full
11	More than five levels of controllees
13	Controllee file specified in erlfn cannot be found
14	Insufficient time to run controllee specified in erlfn
15	System error; restart failed
17	Controllee file specified in erlfn is not executable
18	Mass storage error for file specified in erlfn
19	Error in controllee file or drop file input/output connector specified in erlfn
>19	System error with return word formatted as in STATUS call of SRM

All of the above errors indicate checkpoint or restart failure.

RESTART

A checkpointed task is restarted through a control statement that names the file containing checkpoint information. The system restarts all controllees in response to the single call. Once the controllee chain is reestablished, the system returns control to the checkpointed program at the statement immediately following the checkpoint call.

Restart verifies that all files open at checkpoint time still exist and still occupy the same file space. The file OUTPUT is recreated if necessary. Any tape files attached at checkpoint time are not reattached by restart, however. The restarted task is responsible for reestablishing connection with any tape and for repositioning as required.

A task containing several calls to checkpoint with a given file name might run into conflict during the restart phase. If the restart file is itself named lfn, it cannot call checkpoint with the same lfn. For this case, it is recommended that a switch to another name of the restart file be done before commencing its execution. If this is not done, however, the checkpoint files will still be created using the intermediate checkpoint file names.

MAGNETIC TAPE SUBROUTINES

The magnetic tape subroutines provide a means to write or check tapes labeled according to American National Standard X3.27-1969, Magnetic Tape Labels for Information Interchange. They also can be used to read or write any unlabeled tape. The routines can process labeled multivolume files and multifile volumes. The subroutines can be called from FORTRAN or from the CYBER 200 assembler language using the common calling sequence.

Any application program intended to become a public file for use as a general system utility should be programmed using these subroutines. The subroutines provide a standard interface for communication with an operator who has access to tape units. Messages displayed for the operator describe the proper tape to be mounted; messages might also describe conditions that require operator decisions. In the latter instance, both the error message and the operator response are reported to the dayfile of a batch job or to the terminal of a program executing interactively. The operator also has the ability to abort a tape request at any time. Any action that causes abnormal task termination causes the contents of the associated file information table to be displayed along with the contents of any volume or file labels.

The eight magnetic tape subroutines and their functions are:

Name	Function
Q8ATCHTP	Attach caller to a tape unit. Check or write any volume header label. Logically open a tape volume.
Q8OPENTP	Logically open a tape file. Check or write any file header label.
Q8READTP	Read data from tape.
Q8WRITTP	Write data to tape.
Q8CLOSTP	Close a tape file. Process any file trailer label.
Q8UNLDTP	Close a tape volume and the tape unit for use by the system.
Q8ASGNTP	Assign tape volume by volume serial number; valid only for a multivolume file.
Q8BUFFTP	Assign the virtual address of buffers.

The Q8BUFFTP routine, if used at all, must be the first call. Otherwise, Q8ATCHTP is the first call required to have a tape volume mounted. A call to Q8OPENTP must follow to logically open the tape before Q8READTP or Q8WRITTP can be called to read or write the tape. The last call that references a file should be Q8CLOSTP.

When a file is opened, the system assigns buffers to be used for the subsequent explicit input/output. In response to calls for tape read, physical records are transferred to the buffer in main memory and then a single record is moved to the record location specified in the Q8READTP call. At programmer option, however, the transfer from the buffer to another array can be eliminated and the program can manipulate data directly within the buffer. A call to Q8BUFFTP is required to establish buffer location. A similar procedure can be used with write operations.

A file is referenced in the magnetic tape subroutines by a file name associated with a volume through the Q8ATCHTP call. This name does not appear in the tape label and can be arbitrarily assigned. If the volume contains more than one file, all files in the volume can be referenced by the same name.

TAPE STRUCTURES

Tapes to be read or written by these routines can have any of four structures: binary, BCD, ASCII 6-bit characters (64-character subset), or ASCII 8-bit characters.

For BCD format tape and ASCII 6-bit character tape, each physical tape record is treated as a single logical record. Records can be variable length, to the maximum size established by the buffer. An end-of-file on an existing tape is indicated by the presence of a tape mark or an EOF1 label.

For a tape with data in 8-bit ASCII characters with ASCII control characters, a single call to Q8READTP deals with a single record defined by the ASCII unit separator (#1F). Only the data between two unit separators is returned to the array specified in the call. Similarly, in response to a Q8WRITTP call, the system takes the number of characters specified in the call, adds the unit separator, and transfers the record to the buffer. An end-of-file on an existing tape is indicated by the presence of a tape mark or the ASCII file separator (#1C).

Labels supported by these routines are VOL1, HDR1, EOF1, and EOV1. File header labels HDR2 through HDR9 and user labels UVL, UHL, and UTL are allowed on input tapes, but are skipped by the tape subroutines. Parameters in the subroutine calls establish label field contents to be checked or written. Table 6-1 shows the contents of labels processed.

Characters allowed in the labels are the digits 0 through 9, letters A through Z, and the characters $ " % & ' ( ) * + , - . / : ; < > ? @ # and space. In the descriptions that follow, these characters are known as ANSI label characters.

ANSI labels for 7-track tape are recorded in BCD mode at the density specified for the file. For 9-track tape, ANSI labels are recorded in odd parity at the density specified for the file. Both 7-track and 9-track tape labels are 80 characters long.

SUBROUTINE CALLS

The formats of the magnetic tape subroutine calls are described in the following figures, with notes describing any differences required for assembler language call. In FORTRAN, the calls follow standard conventions in which a parameter can be established by a constant or a variable containing the constant value.

All numeric parameters must be full-word integers. Values for characters in label fields can be specified as character constants or variables containing Hollerith constants left-justified and blank filled to the length required by the field. If the tape is unlabeled, parameters referencing label fields should be set to 0.

Calls are presented below in alphabetical order.

Q8ASGNTP Subroutine Call

The Q8ASGNTP subroutine requests a subsequent volume for a multivolume file or file set. It must be used when Q8WRITTP or Q8READTP senses end-of-tape. Otherwise, a following read or write request causes the task to be aborted.

Q8ASGNTP format is shown in figure 6-2.

For an output tape, the next volume can be specified by volume serial number. Alternatively, a request can be made for any available scratch tape.

For labeled input tape, the system checks that the next volume in sequence has been assigned by the operator. If the assigned volume is out of order, the condition is reported to the program for subsequent processing or unloading.

CALL Q8ASGNTP (lfn, vsn, unit, stat)

lfn — Logical file name to be assigned to a new volume

vsn — Volume serial number of new reel. Must be 1 through 6 ANSI label characters left-justified and blank filled.

If set to $ left-justified and blank filled, any available scratch tape can be assigned.

unit — Variable to which the system returns the unit assigned by the operator.

stat — Variable to which the system returns error code:

0	Volume assigned with no error
1	Format or parameter error
2	File is not waiting for a new volume; call issued at an inappropriate time
3	Reel not available and operator killed tape request
4	Write access requested for tape with unexpired label and operator killed tape request
5	Requested labeled tape does not have VOL1 label and operator killed tape request
6	Volume out of sequence
8	Density of input volume is different from that specified in request
9	Tape request aborted by the operator
A	No write ring for the output file and the operator killed the tape request
B	Ring not allowed on a read only tape and the operator killed the tape request
C	Tape unit is not ready and the operator killed the tape request
D	Read parity error unrecoverable, could not verify scratch tape and the operator killed the tape request

Figure 6-2. Q8ASGNTP Subroutine Format

TABLE 6-1. ANSI LABEL FORMATS

Label	Character Position	Field	Name	Length	Contents	Default Written
Volume header	1 to 3	1	Label Identifier	3	VOL	VOL
	4	2	Label Number	1	1	1
	5 to 10	3	Volume Serial Number	6	Any characters	
	11	4	Accessibility	1	Space	Space
	12 to 31	5	Reserved	20	Spaces	Spaces
	32 to 37	6	Reserved	6	Spaces	Spaces
	38 to 51	7	Owner ID	14	Any characters	Spaces
	52 to 79	8	Reserved	28	Spaces	Spaces
	80	9	Label Standard Level	1	1	1
First file header	1 to 3	1	Label Identifier	3	HDR	HDR
	4	2	Label Number	1	1	1
	5 to 21	3	File Identifier	17	Any characters	Spaces
	22 to 27	4	Set Identification	6	Any characters	Volume serial number of first reel of the set
	28 to 31	5	File Section Number	4	4 digits indicating number of volume in file	0001
	32 to 35	6	File Sequence Number	4	4 digits indicating number of file in multifile set	0001
	36 to 39	7	Generation Number	4	(Not used by the operating system)	Spaces
	40, 41	8	Generation Version Number	2	2 digits indicating the edition of the file	00
	42 to 47	9	Creation Date	6	Space followed by 2 digits for year, 3 digits for day	Current date is used
	48 to 53	10	Expiration Date	6	Same as field 9	Same as field 9
	54	11	Accessibility	1	Any characters	Space
	55 to 60	12	Block Count	6	Zeros	Zeros
	61 to 73	13	System Code	13	Any characters	Spaces
	74 to 80	14	Reserved	7	Spaces	Spaces
First end-of-file	1 to 3	1	Label Identifier	3	EOF	EOF
	4	2	Label Number	1	1	1
	5 to 54	3 to 11	Same as corresponding HDR1 label fields			

TABLE 6-1. ANSI LABEL FORMATS (Contd)

Label	Character Position	Field	Name	Length	Contents	Default Written
First end-of-file (contd)	55 to 60	12	Block Count	6	6 digits indicating number of data blocks since last HDR label group	
	61 to 80	13, 14	Same as corresponding HDR1 label fields			
First end-of-volume	1 to 3	1	Label Identifier	3	EOV	EOV
	4	2	Label Number	1	1	1
	All other fields are identical to EOF1 label.					

Q8ATCHTP Subroutine Call

The Q8ATCHTP subroutine defines the tape characteristics and results in a physical tape being assigned to the task. In response to the call, the system connects with a physical tape unit. If the unit is not ready, or if the tape mounted does not reflect that requested, appropriate messages are displayed for operator intervention to correct the problem. If the operator cannot correct the problem, a code reflecting the problem is passed back to the subroutine.

Parameters in the call identify whether a 7-track or 9-track tape is required, its density, and whether or not the tape should have a write enable ring. The density of an output tape is set according to the parameter in the subroutine call, but a density of 0 indicates that the density was to have been set by the operator or by a previous task. A labeled input tape always is processed at the density used to create the tape, even if it differs from the density specified in the Q8ATCHTP call.

A vsn parameter is required in the call to identify the tape volume to be assigned. The parameter can specify a volume serial number in a label field or simply the sticker number from an unlabeled tape. Any scratch tape can also be requested.

Q8ATCHTP format is shown in figure 6-3. An example in which a labeled input tape is requested is shown in figure 6-4. The tape is assumed to be on a 9-track drive and recorded at 800 bpi density. When the volume label is processed, the owner identification field from the label is returned to variable ONR. The first IF statement in the example allows the tape to be read even if the density is other than the 800 bpi anticipated. Any other error causes a transfer to label 200 where an error message is written to the file OUTPUT.

CALL Q8ATCHTP (acs, lfn, unit, dt, density, vsn, owner, rec, label, stat)

acs — Access mode:

- 0 — Input; read only
- 1 — Output; write ring required
- 2 — Access determined by Q8OPENTP call

lfn — Name by which file is referenced in subsequent calls. Must be 1 through 8 letters or digits beginning with a letter and must be left-justified and blank filled.

unit — Variable to which the system returns the unit assigned by the operator.

dt — Device type:

- 7 — 7-track tape required
- 9 — 9-track tape required
- 0 — Either 7-track or 9-track tape for an output tape. The system returns 7 or 9 to indicate the type of tape assigned by the operator.

density — Tape density:

- 200 — 200 bpi (7-track tape)
- 556 — 556 bpi (7-track tape)
- 800 — 800 bpi (7-track or 9-track tape)
- 1600 — 1600 bpi (9-track tape)
- 0 — Use density set by operator. The system returns the actual density.

Figure 6-3. Q8ATCHTP Subroutine Format (Sheet 1 of 2)

vsn	Volume serial number of reel. Must be 1 through 6 ANSI label characters left-justified and blank filled.
owner†	Value of owner identification field of VOL1 label. Must be 1 through 14 ANSI label characters left-justified and blank filled to 14 characters. Default is all blanks.
rec	Indicator for handling of read parity error or lost data error:
	0 Abort task if data transfer error occurs
	1 Return status to variable indicated by stat parameter but do not abort
label	Indicator for label:
	0 Volume has ANSI labels
	1 Volume is unlabeled
stat	Variable to which the system returns error code:
	0 Tape assigned with no error
	1 Parameter or format error
	2 File already exists
	3 Reel not available and operator dropped tape request
	4 Write access requested for tape with unexpired label and operator killed tape request
	5 Requested labeled tape does not have VOL1 label and operator killed tape request
	6 No input/output connector available
	7 Input reel is not the first in a multivolume set
	8 Density of input volume is different from that specified in request
	9 Tape request aborted by the operator
	A No write ring for the output file and the operator killed the tape request
	B Ring not allowed on a read only tape and the operator killed the tape request
	C Tape unit is not ready and the operator killed the tape request
	D Could not verify the scratch tape and the operator killed the tape request

†Note: The owner field must start on a byte boundary.

Figure 6-3. Q8ATCHTP Subroutine Format (Sheet 2 of 2)

```
      MODE=0
      LFN=8HFILEA
      VSN=6HU12345
      CALL Q8ATCHTP(MODE,LFN,IUNIT,9,800,
     +VSN,ONR,0,0,ISTAT)
      IF(ISTAT.EQ.8) ISTAT=0
      IF(ISTAT.GT.0) GO TO 200
      •
      •
      •
200 PRINT(1H1,"TAPE NOT ASSIGNED")
```

Figure 6-4. Example of Q8ATCHTP Use

Q8BUFFTP Subroutine Call

The Q8BUFFTP subroutine is an optional routine that specifies the location to be used for input/output buffers. It presumes that the program manipulates the buffer contents and calls Q8READTP or Q8WRITTP with the recloc parameter set to 0. If used, Q8BUFFTP must be the first file reference.

Q8BUFFTP format is shown in figure 6-5. Two buffers need not be specified. Buffer 1 is the first buffer accessed.

Virtual space for the buffer is mapped during Q8ATCHTP processing and must not be mapped by any other program action.

CALL Q8BUFFTP (lfn, buf1, buf2, blen, stat)

lfn	Logical file name
buf1	Array specifying first buffer.
buf2	Array specifying second buffer.
blen	Number of 8-bit bytes in each buffer.
stat	Variable to which the system returns error code:
	0 Buffer assigned with no error
	1 Format or parameter error
	2 File name is already active
Note:	Buffer location must be aligned on a word boundary and be specified as a virtual bit address.

Figure 6-5. Q8BUFFTP Subroutine Format

In the absence of a Q8BUFFTP call, the system assigns tape buffers at virtual bit address 1 000 000 000 + ((n x 3012) x 512), where n is the number of buffers.

Q8CLOSTP Subroutine Call

The Q8CLOSTP subroutine closes a file. It is required before a subsequent file of a multifile volume can be opened. It must be the last reference to a file except for a request for volume unload. The subroutine processes any end-of-file label.

Q8CLOSTP format is shown in figure 6-6.

CALL Q8CLOSTP (lfn, func, blkcnt, stat)

lfn — Logical file name

func — File positioning and disposition:

- 0 Rewind and unload
- 1 No rewind and position to beginning of next file

blkcnt — Variable to which system returns the number of blocks, excluding labels or tapemarks, read or written.

stat — Variable to which system returns error code:

- 0 Close completed with no error
- 1 Format or parameter error
- 2 Bad lfn
- 3 Block count error

Figure 6-6. Q8CLOSTP Subroutine Format

For input files, a block count check occurs when the file is closed at its end. A close before end-of-tape is sensed, however, inhibits this check. An end-of-tape condition can occur when an input file is closed with no rewind. In this instance, the condition indicates that the file is continued on another volume. To properly close the file, the succeeding volume must be assigned through a call to Q8ASGNTP and Q8CLOSTP reissued.

A close with no rewind positions a tape at the beginning of the next file.

Q8OPENTP Subroutine Call

The Q8OPENTP subroutine logically opens a file for processing. A Q8ATCHTP call must precede the Q8OPENTP call in order to associate the volume with the task. The tape is assumed to be positioned at the beginning of a file header label or after a tape mark of an unlabeled tape.

For a labeled tape, the HDR1 label is processed in response to this call. Specified fields are written to the label of an output tape or checked against the actual fields in an existing label. For an unlabeled tape, label field parameters should be set to 0.

The access parameter determines whether a file is to be read or written. If the volume is declared in the Q8ATCHTP call as having input/output access, a Q8OPENTP parameter specifying output prohibits any subsequent read on the volume.

Q8OPENTP format is shown in figure 6-7. Figure 6-8 shows an example in which a labeled output 9-track 1600 bpi tape is written with a volume serial number of U782. The owner identification field is set to the characters USER # 253700 . File structure is defined as binary records 4096 bytes in length.

CALL Q8OPENTP (acs, lfn, fileid, setid, struct, buf, ret, gn, vn, stat)

acs — Access mode:

- 0 Input; read only
- 1 Output; write only

lfn — Logical file name

fileid — For a file opened for output, value to be written in file identifier field of HDR1 label. Must be 1 through 17 ANSI label characters blank filled to 17 characters. Default is spaces.

For a file opened for input, file identifier to be checked against label field. The system returns the field read from the label if it is different from that specified or if the call is issued with this parameter set to all spaces.

setid — For a file opened for output, value to be written in set identification filed of HDR1 label. Must be 1 through 6 ANSI label characters blank filled to 6 characters. Default is spaces.

For a file opened for input, set identification to be checked against label field. The system returns the field read from the label if it is different from that specified or if the call has been issued with this parameter set to all spaces.

struct — File structure:

- 0 BCD characters
- 1 Binary data
- 2 6-bit characters of 64-character ASCII subset
- 8 8-bit ASCII characters including ASCII unit separators

buf — Number of 8-bit bytes in largest/physical record on tape. Must be 4 through 98304.

ret — For an output file, number of days file is to be retained. Must be 0 through 999 days. The system converts this number to an expiration date for the HDR1 label.

Figure 6-7. Q8OPENTP Subroutine Format (Sheet 1 of 2)

For an input file, a full word containing the creation date in the upper 32 bits and the expiration date in the lower 32 bits as read from the file label.

gn — For an output file, generation number 1 through 9999 to be written in the HDR1 label.

For an input file, generation number read from the file label.

vn — For an output file, version number 0 through 99 to be written in the HDR1 label.

For an input file, version number read from the file label.

stat — Variable to which the system returns error code:

0 Normal completion with no error
1 Format or parameter error
2 Bad file name
3 File already open
4 Read or write access requested does not match that allowed by Q8ATCHTP call
5 Attempt to write unexpired file
6 File identifier specified does not match existing label
7 Set identification specified does not match existing label
8 End-of-information reached
9 End-of-tape

Figure 6-7. Q8OPENTP Subroutine Format (Sheet 2 of 2)

```
 OWNER='USER # 253700 '
 SETID='BACKUP'
 CALL Q8ATCHTP (1,'FILEA',IUNIT,9,1600,'U782',
+OWNER,0,0,ISTATUS)
 IF(ISTATUS.GT.0) GO TO 400
 CALL Q8OPENTP(1,'FILEA','FILEA',SETID,1,4096,
+30,1,0,ISTATUS)
```

Figure 6-8. Example of Q8OPENTP Use

The tape retention parameter in the call becomes an expiration date field in the file header label. The five-digit date has the format yyddd representing the year and the day of the year.

Q8READTP Subroutine Call

The Q8READTP subroutine passes records from the active tape buffer to an array. It also controls the initiation of buffer read operations.

Q8READTP format is shown in figure 6-9.

If an end-of-tape condition is sensed during execution, a status code is returned to the program. In this instance, the program must issue a Q8ASGNTP request for a subsequent volume; otherwise, another Q8READTP request causes the job to be aborted. After a new volume has been assigned, Q8READTP must be called again to complete the read interrupted by the end-of-tape condition.

The information returned is affected by the tape structure:

For 6-bit ASCII and BCD format, each read request returns the entire physical record.

For ASCII-8 format, each read request returns data between two unit separators, but the separators themselves are not returned. An end-of-file status is returned when read encounters a tape mark or a file separator. If a record is longer than #500 bytes, it is returned as two or more records.

CALL Q8READTP (lfn, recloc, rl, stat)

lfn — Logical file name

recloc — Array to receive record read.

If this parameter is set to 0, the record is read into the buffer but is not transferred from the buffer. The location in the buffer is returned to this parameter.

rl — Variable for record length:

For coded files, must be 0. The system returns the number of bytes read.

For binary files, number of bytes to be read.

stat — Variable to which the system returns error code:

0 Normal completion with no error

1 Format or parameter error

2 Bad lfn

3 Read access denied for this buffer

4 Physical record is larger than buffer length and record has been truncated to length specified by rl

5 Lost data; hardware failure or the physical record does not contain an integral number of characters and the last partial character is lost

6 Tape, SBU, or transmission parity error

7 End-of-file

8 End-of-tape encountered

Figure 6-9. Q8READTP Subroutine Format

For binary format, each read request returns the number of 8-bit bytes specified in the request.

At the completion of the read request, the number of characters read is returned to the rl parameter. If the physical record is larger than the buffer size, it is truncated.

When binary records are being read, the read request can specify a number of characters which is longer or shorter than a physical record. In this instance, SRM institutes as many tape read functions as necessary to fill the record area defined. This technique can be used to read an entire tape with a single Q8READTP call.

Errors encountered during the read are handled as specified by the rec parameter of the Q8ATCHTP call for the file. If rec is set to 1:

Read parity errors are reported by a status code, but the record is returned.

Lost data errors are reported by a status code and rl is set to 0.

Hardware errors resulting in the entire record being lost abort the task.

If rec is set to 0, any of these conditions aborts the task.

Q8UNLDTP Subroutine Call

The Q8UNLDTP subroutine causes a tape volume to be rewound and unloaded. The unit is also released from the task and made available to the system for reassignment.

The routine is not required, as the same function can be performed with the Q8CLOSTP call that closes the file. The volume unload is not performed unless a Q8CLOSTP has previously closed any open file on the volume.

Q8UNLDTP format is shown in figure 6-10.

CALL Q8UNLDTP (lfn, stat)

lfn — Logical file name

stat — Variable to which the system returns error code:

0	Tape unloaded with no error
1	Bad lfn
2	File open on this volume

Figure 6-10. Q8UNLDTP Subroutine Format

Q8WRITTP Subroutine Call

The Q8WRITTP subroutine passes a record from an array to the active tape buffer. It also controls the initiation of buffer write operations.

Q8WRITTP format is shown in figure 6-11.

If an end-of-tape condition is sensed during execution, a status code is returned to the program. In this instance, the program must issue a Q8ASGNTP request for a subsequent volume; otherwise, another Q8WRITTP request causes the job to be aborted. After a new volume has been assigned, Q8WRITTP must be called again to complete the write interrupted by the end-of-tape.

CALL Q8WRITTP (lfn, recloc, rl, stat)

lfn — Logical file name

recloc — Array from which record is to be written.

If this parameter is set to 0, the record is written from the buffer directly.

rl — Number of bytes to be written.

stat — Variable to which the system returns error code:

0	Normal completion with no error
1	Format or parameter error
2	File not open
3	Write access denied for this buffer
4	Record exceeds buffer size
5	End-of-tape with write completed
6	End-of-tape with write in process

Figure 6-11. Q8WRITTP Subroutine Format

The information written is affected by the tape structure:

For 6-bit ASCII and BCD format, each write request writes the number of characters specified as a physical record.

For ASCII-8 format, a unit separator (#1F) is appended to the number of characters specified before being moved to the buffer. The last block written to the file is padded with #1C if necessary.

For binary format, each write request writes the number of 8-bit bytes specified in the request.

If the number of characters specified is greater than the buffer size, the record is written as more than one physical record.

A permanent hardware error aborts the task.

SYSTEM RECORD MANAGER

System Record Manager (SRM) is an interface between a user program and the system messages involved with file input/output. For the most part, it relieves the user program from the need to deal with the details of the Alpha and Beta words of the system messages. Once the user provides SRM with various parameters, SRM constructs the proper message and issues it to the system. Examples of such messages are those that create or destroy files, those that open, close, or reduce file length, and those that perform explicit input/output. Note that SRM nonprivileged functions will work only with local files or attached permanent files.

SRM also provides logical blocking and deblocking capabilities that allow a program to deal with logical, rather than physical, records. Blocking and deblocking can be performed on files in SRM-structured format and files of ASCII data in unstructured format. As described in section 3, SRM-structured files contain control words,

while ASCII data files contain ASCII control characters. In both types of file, SRM can detect three levels of logical file boundaries:

SRM-Structured File	ASCII File
Record	Unit
Section	Group
File	File

The type of file (SRM-structured or unstructured) affects the calls that can be used with the file.

When an unstructured or structured file is to be processed, SRM automatically sets the input/output mode to either explicit or implicit as established by the mode field in the FIT. (Previously set by the user.) The only additional responsibility the user has during explicit input/output is to define a buffer and to open it with a BUFOP function. A file open for explicit input/output is read or written through the functions READ and WRITE. If the file has been opened for implicit input/output, the file is accessed by referencing the virtual space associated with the file.

Table 6-2 summarizes the functions available through SRM.

FILE INFORMATION TABLE

To facilitate input/output processing, System Record Manager maintains certain information for each file in a file information table (FIT), as shown in figure 6-12.

A file is identified to SRM by the location of its FIT. Each FIT is allocated and maintained internally by SRM. All user references to the FIT use an internal file name (IFN). When a user call is made to SRM, IFN is used to locate the relative position of a FIT within a predefined common block which is defined and maintained by SRM. The IFN associated with each FIT is provided by the user.

The SRM functions GENFIT and GENFITX establish a file information table proper for implicit or explicit mode, respectively. The mode is used when the file is created or opened for processing; and, once the file is opened, the appropriate SRM functions must be called to access the file. GENFIT or GENFITX must be used before any other file function.

The function SETFIT can be used to set a particular FIT field. The function GETFIT retrieves a value from the table.

All common blocks allocated by SRM have names prefixed with two characters which define the type of common block. Q9 represents labeled common blocks while 99 represents numbered common blocks. Q9FITMAN is the first common block used to provide virtual space in which FITs are generated.

Directory information which is used internally by SRM to locate FITs is also embedded in Q9FITMAN. A directory is made up of one-word entries containing the IFN of each active FIT. The position of an IFN in the directory determines the location of the FIT; the FIT associated with the first directory entry immediately follows the directory. Currently, the directory has a length of 100 words which represents the maximum number of FITs that can be generated in a single task. Once a FIT is generated, it remains defined for the entire execution.

ERROR PROCESSING

During execution of SRM functions, various errors might occur. Such errors are reported via the error exit address in the FIT. The field can be set with a GENFIT, GENFITX, or SETFIT function. It is important that an error exit address exist at all times. This insures proper reporting of errors.

An error exit address must be set to return to the same module which issues the SRM function. When an error exit is taken, the data base of the calling routine is restored. If the error exit address is in a different module, the wrong data base is loaded.

Once an error exit is taken, it is the programmer's responsibility to determine the cause of the error. The STATUS function returns error information pertaining to the previous SRM function.

LANGUAGE DIFFERENCES

Several means can be used to call the SRM functions, depending on the program language. Default values exist for parameters only when the functions are called with a macro.

FORTRAN Subroutine Calls

The FORTRAN subroutine calls have the general format:

```
CALL Q7name (parameters)
```

The names by which they are called are the macro names in table 6-2 prefixed by the characters Q7, except as noted in the table. The call that generates an FIT for explicit input/output, for instance, is:

```
CALL Q7GNFITX (parameters)
```

IMPL Subroutine Calls

The IMPL subroutine calls have the general format:

```
CALL   Q8name(parameters)
```

The names by which they are called are the macro names of table 6-2, prefixed by the characters Q8. Unlike FORTRAN, there are no exceptions. The same call, illustrated above in FORTRAN, would appear in IMPL as:

```
CALL   Q8GENFITX(parameters)
```

TABLE 6-2. SYSTEM RECORD MANAGER FUNCTIONS

STATUS Function Code	Macro Name	Applicable File or I/O Type	Function
		File Information Table Manipulation	
900	GENFIT	Implicit I/O	Generate file information table for implicit I/O.
901	GENFITX†	Explicit I/O	Generate file information table for explicit I/O.
902	GETFIT	Any	Retrieve value of specified FIT field.
903	SETFIT	Any	Set value of specified FIT field.
		File Initialization and Termination	
90A	OPEN	Any	Connect file to program. Issue OPEN FILE message.
90C	CLOSE	Any	Disconnect file from program. Issue CLOSE FILE message.
		Data Transfer and Structuring	
911	GET	Structured	Begin record transfer to single location.
915	GETL	Structured	Begin record transfer to more than one location.
912	GETM	Structured	Transfer same record data to single location.
916	GETML	Structured	Distribute same record data to several locations.
913	PUT	Structured	Begin writing a record from a single location.
917	PUTL	Structured	Begin writing a record gathered from several locations.
914	PUTM	Structured	Continue writing same record from a single location.
918	PUTML	Structured	Continue gathering data for same record from several locations.
919	GETBCD	Unstructured	Transfer single unit of ASCII file data.
91A	PUTBCD	Unstructured	Write single unit of ASCII file data.
91B	WEOR	Structured	Write end-of-record.
91C	WEOS	Structured	Write end-of-section.
91D	WEOI	Structured	Write end-of-information.
90E	BUFOP	Explicit I/O	Define or release buffer.
90F	READ	Explicit I/O	Transfer data directly.
910	WRITE	Explicit I/O	Transfer data directly.
		File Positioning	
91E	REWIND	Any	Position at beginning-of-information.
91F	SKIP	Structured	Skip forward or backward by records.
920	SKIPS	Structured	Skip forward or backward by sections.
921	BKSPC	Unstructured	Backspace one unit of ASCII file data.

TABLE 6-2. SYSTEM RECORD MANAGER FUNCTIONS (Contd)

STATUS Function Code	Macro Name	Applicable File or I/O Type	Function
		File Control at Message Interface	
904	CREATE	Mass storage	Make file known to system. Issue CREATE FILE message.
906	DESTROY††	Any	Sever connection with program. Issue DESTROY FILE message.
922	GIVE	Mass storage	Change file owner. Issue GIVE FILES message.
908	REDUCE	Mass storage	Reduce file length. Issue REDUCE FILE message.
909	CHANGE	Mass storage	Change file name or account. Issue CHANGE FILE message.
		Miscellaneous	
925	STATUS	Any	Check input/output activity. Issue GIVE UP CPU message.
924	TPFCN	Tape	Perform miscellaneous tape functions.
923	TERM	- -	Terminate program execution. Issue TERMINATE message.
		Privileged User	
905	PCREAT	Any	Privileged create.
907	PDESTR	Any	Privileged destroy.
90B	POPEN	Any	Privileged open.
90D	PCLOSE	Any	Privileged close.

†Called in FORTRAN as Q7GNFITX.

††Called in FORTRAN as Q7DESTR.

Assembler Language Macros

The SRM functions can be accessed by the names shown in table 6-2. The macros generate subroutine calls to SRM modules. To access the macros, an assembler language program must contain an LIBP statement referencing the file SRMMACRO. Further, the external subroutines which the macros call must be declared in an EXTC statement. SRM generates necessary EXTC statements in the user's data base. Each EXTC which is generated is labeled with labels having the format:

Q8SRMnn

where nn are the last two digits of the function code in table 6-2.

In the course of execution, the user's data base registers are saved but the parameter registers #F8 through #FF are destroyed. The temporary work registers #0C through #13 are also used. Consequently, when SRM macros are called, the calling routine should not use registers #F8 through #FF or #0C through #13 for storage purposes.

In a macro call, a parameter can designate a register containing the value of a variable, or it can be a constant. The following forms of SETFIT macro are all valid:

```
1.  IFN         GEN     "FILE1"
    KEYWORD     GEN     "LFN"
    FILENAME    GEN     "TEST"
                  .
                  .
                  .
                MSEC    2
                SETFIT  IFN,KEYWORD,FILENAME

2.  IFN         RDEF    "FILE1"
    KEYWORD     RDEF    "LFN"
    FILENAME    RDEF    "TEST"
                  .
                  .
                  .
                MSEC    2
                SETFIT  IFN,KEYWORD,FILENAME

3.              SETFIT  "FILE1","LFN","TEST"
```

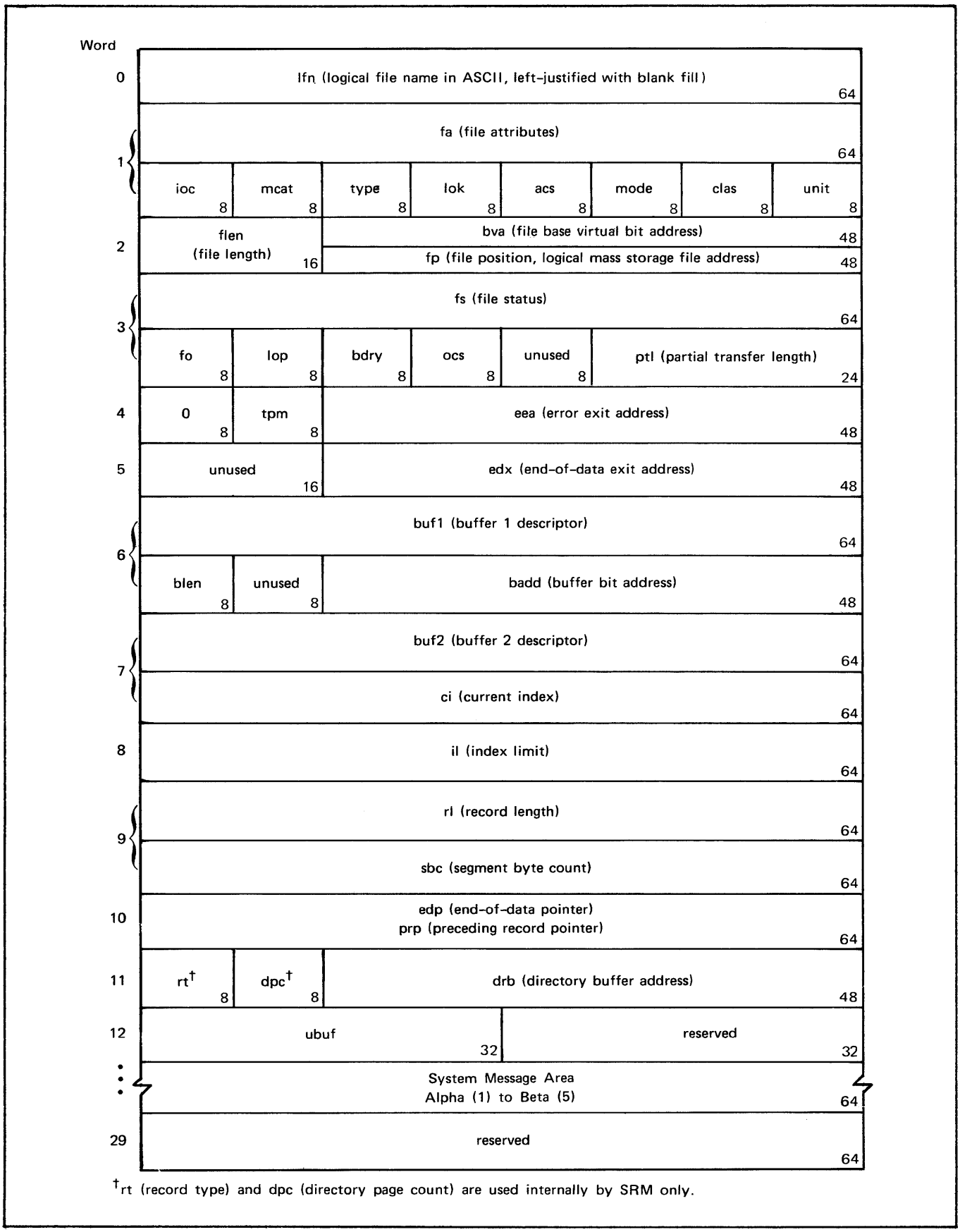


Figure 6-12. System Record Manager File Information Table Format

```
4.          SETFIT  FILE1,LFN,TEST
```

In the fourth example, the parameters are taken as constants provided that they have not been defined in data or common sections.

Assembler Language Subroutines

The calls for the assembler language subroutines have the general format:

```
name_
```

where name is the macro name shown in table 6-2.

Conventions for registers and parameters are:

Register #FF	First parameter
Register #FE	Second parameter
.	.
.	.
.	.
etc.	Nth parameter
Register #17	Parameter count in bits 0 through 15

Therefore, for a create, a program might contain parameters IFN, PACKID, and FRAG in registers #FF through #FD, the data base in register #1E, and code:

```
RTOR   IFN,REGFF          *PARAMETER 1
RTOR   PACKID,REGFE       *PARAMETER 2
RTOR   FRAG,REGFD         *PARAMETER 3
RTOR   CREATE_DB,REG1E    *DATA BASE
ELEN   #17*64,3           *PARAMETER
                          COUNT
BSAVE  REG1A,CREATE_      *CALL CREATE_
```

FUNCTION CALLS

The SRM functions, given in their FORTRAN form, are described below in the following order:

File utility functions create, destroy, change, reduce, give.

Alphabetical listing of subroutines.

File Utility Function Calls

Five SRM subroutines issue messages that correspond to capabilities available through system utilities. They are:

Subroutine	Use
CHANGE	Issue CHANGE FILENAME OR ACCOUNT message to change the name of a file or to change the account number associated with an existing file.
CREATE	Issue CREATE FILE message to establish pack or tape volume on which a file is to reside. Mass storage files created are always local files.
DESTROY	Issue DESTROY FILE message to sever connection between a program and a local file or attached permanent file. For a local file, mass storage space is released. For a permanent file, the file is detached.
GIVE	Issue GIVE FILE message to change owner of file.
REDUCE	Issue REDUCE FILE LENGTH message to reduce length of an existing private file. A file can be reduced only when it is not open.

These subroutines can be used in any program. Two other subroutines, PCREAT and PDESTR, are available for use by privileged tasks to create or destroy a file associated with a user number other than that under which the task is executing. Format of these subroutines is shown in figure 6-13.

BKSPC Call

The BKSPC subroutine is valid only for an unstructured file containing ASCII data with ASCII control characters. It causes the file to be backspaced one record defined by the ASCII unit separator (#1F).

BKSPC format is shown in figure 6-14.

If the file is positioned at beginning of information or at beginning of tape volume, SRM takes the end-of-data exit specified in the FIT. If the file is positioned at the first character of a record, the file is skipped backward and positioned at the first character of the preceding record.

If the file is positioned somewhere other than the first character of a record, it is skipped backward to the first character of the same record where the file was initially positioned.

Each of the following ASCII control characters is considered as a record in itself: FS (#1C), GS (#1D), RS (#1E).

BUFOP Call

The BUFOP subroutine is valid only for a file being accessed by explicit input/output. It opens or closes one of two file buffers.

The buffer close operation writes out any information in the buffer to the file storage device. BUFOP should be called before file close.

BUFOP format is shown in figure 6-15.

CALL Q7CHANGE (ifn, new, c)

ifn — Internal name of FIT.

new — When c is 0, new file name in ASCII, left-justified and blank filled. When c is 1, new account number in ASCII, right-justified.

c — Indicator of item to be changed:

- 0 Change file name.
- 1 Change account number.

CALL Q7CREATE (ifn, packid, 0.)

packid — Variable containing identifier of pack or tape on which file is to be created, formatted as:

Bits	Contents
0-15	#0000.
16-63	1-6 character pack identifier or tape volume serial number.

If packid is 0, the system selects a pack.

CALL Q7DESTROY (ifn, owner)

owner — Ownership category:

- 0 Private.
- 1 Public.
- 2 Pool.

CALL Q7GIVE (ifn, userno)

userno — User number to receive file.

CALL Q7REDUCE (ifn, len)

len — Number of small pages in new file length.

CALL Q7PDESTR (ifn, userno)

CALL Q7PCREAT (ifn, filei, packid)

filei — Array containing a 14-word correctly formatted image of the file index table entry.

Figure 6-13. CHANGE, CREATE, DESTROY, GIVE, REDUCE Formats

CALL Q7BKSPC (ifn)

ifn — Internal name of FIT.

Figure 6-14. BKSPC Format

CLOSE and PCLOSE Calls

The CLOSE and PCLOSE subroutines sever the connection between the executing program and the specified file.

CLOSE closes a file from a nonprivileged program. It can also be used to change file attributes.

CALL Q7BUFOP(Ifn,opcode)

Ifn — Internal name of FIT.

opcode — Buffer identifier and action to be taken:

- 1 Open buffer 1
- 2 Close buffer 1
- 3 Open buffer 2
- 4 Close buffer 2

Figure 6-15. BUFOP Format

PCLOSE closes a file from a privileged program. It can also be used to destroy the file. It must be used for a file opened by POPEN.

The close operation does not necessarily perform file termination procedures, therefore the following functions should be issued before CLOSE:

For output files in SRM-structured format accessed by implicit input/output, a WEOI function should be issued.

For output files accessed by explicit input/output, a BUFOP function should be issued to empty the buffer.

CLOSE and PCLOSE formats are shown in figure 6-16.

CALL Q7CLOSE (ifn, change)

CALL Q7PCLOSE (ifn, dest)

ifn — Internal name of FIT.

change — Indicator that file attributes and base virtual address are to be changed:

- 0 Do not change.
- ≠0 Change according to fa and bva fields in FIT.

dest — Indicator of file disposition:

- 0 Do not destroy file. Default.
- 1 Close and destroy file.

Figure 6-16. CLOSE and PCLOSE Formats

GENFIT and GENFITX Calls

These subroutines generate and initialize the FIT.

GENFIT initializes an FIT for implicit input/output.

GENFITX initializes an FIT for explicit input/output.

A call to one of these routines must be the first reference to a file.

GENFIT and GENFITX formats are shown in figure 6-17.

CALL Q7GENFIT (ifn, len, bva, fa, eea, fo)

CALL Q7GNFITX (ifn, len, bufd, fa, eea, fo, drb, tpm)

ifn — Internal file name defined by caller and used to identify a particular FIT. The name can be a 64-bit quantity having any value other than #2020202020202020.

len — Length of file in number of small pages. Default value is #40.

bva — Base virtual address for mapping in the file.

fa — File attributes, specified as the name of a set of eight elements corresponding to the eight fields of the Beta(2) word of the CREATE FILE message. The first and last elements in the set should be null. Any other null element selects the default noted.

Element		
1	(ioc)	Set by SRM to #FF so that automatic input/output connector assignment occurs. To force usage of a particular connector, the SETFIT call must be used.
2	(mcat)	Management category: 0 Mass storage file 1 Scratch file 2 Output file 3 Write temporary file 4 Magnetic tape file 5 User-generated drop file 7 Batch file
3	(type)	File type: 0 Physical data file 2 Virtual code file 7 7-track tape 9 9-track tape
5	(acs)	File access allowed: 0,4 No access 1,5 Write access 2,6 Read access 3,7 Write and read access
6	(mode)	Input/output mode: 0 Explicit. 1 Implicit. Default.
7	(clas)	Security access level, 1 through 255: 0 Same level as task ≠0 Level specified
8	(unit)	Logical unit number of pack on which mass storage file resides, or physical unit number of tape unit; to be returned by the system.

eea — Address expression representing error exit virtual address. Default value is 0, which is a fatal error.

fo — File structure:

0 Structured
1 Unstructured

bufd — Buffer descriptor word in the following format:

Bits	Contents
0–7	Number of small pages in buffer.
8–15	Unused.
16–63	Address of buffer where data transfer requests deposit or obtain information.

drb — Parameter designating virtual bit address for directory buffer. The buffer size is preset to one page. The default bit address is determined to be the next page following the data buffer for the file.

tpm — Optional parameter designating tape mode:

0 BCD (7-track tape)
1 Binary (7- or 9-track tape)
2 Binary ASCII (7-track tape)
4 Mass storage file (default)

Figure 6-17. GENFIT and GENFITX Formats

GET and GETL Calls

The GET and GETL subroutines are valid only for SRM-structured files. They transfer data from the beginning of a record to specified program areas.

GET returns record data to a single program location. A parameter of the GET call specifies the number of characters to be transferred.

GETL returns record data to a list of locations. It allows data from a single record to be transferred into separate scattered areas of program space. A parameter of GETL specifies an address of a list; the list itself identifies each area to which data is to be returned and the number of characters to be transferred to that area.

GET format is shown in figure 6-18; GETL format in figure 6-19.

CALL Q7GET (ifn, bc, da, edx, name)

CALL Q7GETM (ifn, bc, da)

ifn	Internal name of FIT.
bc	Number of characters to be read.
da	Array to which data is to be returned.
edx	Name of subroutine to be executed when an end-of-data condition occurs before the requested number of characters have been read.
name	Variable to which a named record name is to be returned, left-justified and blank filled. If record is not named, spaces are returned.

Figure 6-18. GET and GETM Formats

CALL Q7GETL (ifn, pl, edx)

CALL Q7GETML (ifn, pl)

ifn Internal name of FIT

pl Parameter list descriptor word in format:

Bits	Contents
0-15	Number of data descriptors in parameter list
16-63	Address of first word in parameter list

edx Name of subroutine to be executed when an end-of-data condition occurs before the requested number of characters has been read.

Format of each data descriptor in parameter list is:

Bits	Contents
0-15	Number of characters to be transferred from the file
16-63	Address to receive data

Figure 6-19. GETL and GETML Formats

Both GET and GETL always transfer data from the beginning of a record. If a file is positioned in the middle of a record at the time one of these calls is made, SRM skips forward to the next record boundary before the data transfer. Data between the last character read and the end of the record must be accessed by a call to GETM or GETML.

The number of characters transferred in response to a call for GET or GETL is determined by the number of characters specified in the call. Transfer of data terminates after the specified number of characters or after the end of the record is detected, whichever comes first.

GETM and GETML Calls

The GETM and GETML subroutines are valid only after a preceding GET or GETL call or after a preceding GETM or GETML call. They transfer data from the midst of a record; they cannot be used to initiate reading from the beginning of a record.

GETM returns record data to a single program location.

GETML returns record data to a list of locations.

GETM format is shown in figure 6-18; GETML format in figure 6-19.

If an end-of-data condition occurs before all requested data has been transferred, control returns to the end-of-data exit established by the edx parameter of the preceding GET or GETL call.

GETBCD Call

The GETBCD subroutine is valid only for an unstructured file containing ASCII control characters. It transfers the next sequential record for the file to the location specified in the call.

GETBCD format is shown in figure 6-20.

In response to this call, SRM transfers characters from the current file position up to and including the ASCII unit separator control character (#1F). The character count parameter of the GETBCD call establishes the maximum number of characters transferred.

CALL Q7GETBCD (ifn, da, cc)

ifn	Internal name of FIT
da	Array to which record is to be returned
cc	Maximum number of characters to be read

CALL Q7PUTBCD (ifn, da, cc)

da	Array from which characters are to be written
cc	Number of characters to be written

Figure 6-20. GETBCD and PUTBCD Formats

If the first character to be transferred is the ASCII control character for a record, group, or file (#1E, #1D, or #1C), SRM transfers control to the end-of-data exit indicated in the FIT.

OPEN and POPEN Calls

The OPEN subroutine connects the program with an existing mass storage file.

OPEN opens a file for a program executing under a nonprivileged user number.

POPEN opens a file for a program executing under a privileged user number.

One of the parameters of OPEN can be used to change the file type. In this instance, SETFIT should be used to change the value of the TYPE field before OPEN is called. A physical data file or a virtual data file can be established.

OPEN and POPEN formats are shown in figure 6-21. They result in an OPEN FILE system message.

CALL Q7OPEN (ifn, change)

ifn	Internal name of FIT
change	Indicator that file type is to be changed:
	0 Do not change file type
	≠0 Change file type to that indicated in the type field of word 1 of FIT

CALL Q7POPEN (ifn, userno, filei)

userno	User number associated with file
filei	Array of 14 words to which the file index table entry is to be returned

Figure 6-21. OPEN and POPEN Formats

PUT and PUTL Calls

The PUT and PUTL subroutines are valid only for SRM-structured files. They transfer data from a specified program location to the beginning of a new file record.

PUT transfers data from a single program location.

PUTL transfers data from a list of locations. It allows data to be gathered up into a single record.

PUT format is shown in figure 6-22; PUTL format in figure 6-23. The format of the parameter list descriptor word for PUTL is that shown in figure 6-19.

Both PUT and PUTL begin a new record. Depending on the record length information in the call, a whole record might or might not be written. If additional data is to be written to a record begun by PUT or PUTL, the PUTM or PUTML subroutines must be called.

CALL Q7PUT (ifn, bc, da, rl, name)

CALL Q7PUTM (ifn, bc, da)

ifn	Internal name of FIT
bc	Number of characters to be written
da	Array from which data is to be written
rl	Total number of characters in record:
	>0 Total number of characters
name	Name to be written for a named record, left-justified and blank filled. Optional.

Figure 6-22. PUT and PUTM Formats

CALL Q7PUTL (ifn, pl, rl)

CALL Q7PUTML (ifn, pl)

ifn	Internal name of FIT
pl	Parameter list descriptor word in format shown for GETL
rl	Number of characters to be written from all locations in list

Figure 6-23. PUTL and PUTML Formats

For either of these calls, a record length of 0 can be specified. In this instance, SRM adds data to the record in response to PUTM or PUTML calls until such time as the program issues WEOR to end the record.

If a record length is specified other than 0, that length establishes the maximum allowed for the record. Subsequent PUTM or PUTML calls transfer data only to the limit set by PUT or PUTL. The value for the rl in PUT or PUTL should reflect the ultimate record size.

PUTM and PUTML Calls

The PUTM and PUTML subroutines are valid only after a preceding PUT or PUTL call or after a preceding PUTM or PUTML call. They transfer data from a program to a record under construction.

PUTM writes data from a single program location.

PUTML writes data from one or more program locations in the order presented in a parameter list.

PUTM format is shown in figure 6-22; PUTML format in 6-23. The format of the parameter list descriptor word for PUTML is that shown in figure 6-19.

The maximum number of characters that can be written to a record from a series of calls is established by the PUT or PUTL call that begins the record.

PUTBCD Call

The PUTBCD subroutine is valid only for a file in unstructured format containing ASCII data. It transfers the specified number of characters to a file. The program is responsible for file boundaries, including unit separators. SRM does not add ASCII control characters to the record.

PUTBCD format is shown in figure 6-20.

READ Call

The READ subroutine is valid only for a file being accessed by explicit input/output. It transfers data from an SRM-structured file or an unstructured file to a buffer. Parameters of the call determine which buffer is to receive data and whether or not the central processing unit is to be relinquished while the data transfer is taking place.

READ format is shown in figure 6-24.

CALL Q7READ (ifn, bufno, cpu)

CALL Q7WRITE (ifn, bufno, cpu)

ifn — Internal name of FIT

bufno — Number of buffer to be opened. Must be 1 or 2.

cpu — Indicator that control of the central processing unit is to be relinguished. Optional.

- 0 Do not give up CPU. Default.
- 1 Give up control of CPU until request complete.

Figure 6-24. READ and WRITE Formats

REWIND Call

The REWIND subroutine positions a file as follows:

A tape file is positioned at the beginning of the current volume.

A mass storage file is positioned at beginning of information.

Any output file is not rewound until any data in its buffer is written to the file. If the last operation of the file was a write other than WEOI for an SRM-structured file, an end-of-information control word is written before the rewind operation occurs.

REWIND format is shown in figure 6-25.

SKIP Call

The SKIP subroutine is valid only for an SRM-structured file. It positions the file to a record boundary by skipping forward or backward.

CALL Q7REWIND (ifn)

ifn — Internal file name of the FIT

CALL Q7SKIP (ifn, rc)

rc — Number of records to be skipped:

- >0 Skip forward
- 0 Skip to end-of-record
- <0 Skip backward

CALL Q7SKIPS (ifn, sc)

sc — Number of sections to be skipped:

- >0 Skip forward
- 0 Skip to end-of-section
- <0 Skip backward

Figure 6-25. REWIND, SKIP, and SKIPS Formats

SKIP format is shown in figure 6-25. The direction of the skip is determined by the sign of the count parameter. Figure 6-26 illustrates positioning that results from three calls to SKIP, assuming the current position is within record 3.

When the rc parameter is 0, the file is skipped to the next end-of-record control word if the file is positioned in the middle of a record when the call is made. No action is taken if the file is already positioned at a record boundary.

When rc is a positive nonzero value, the file is skipped forward rc full records. If the file is positioned at some point within a record when the call is made, the file is moved to the next end-of-record control word and then skipped forward rc records.

When rc is a negative value, the file is skipped backward. If the file is positioned within a record when the call is made, the file is moved backward until an end-of-record control word is encountered, counting 1 for this action. If the file is at a record boundary, the file is skipped backward the number of full records specified by the negative rc.

Any output file is not repositioned backward until any data in its buffer is written to the file.

SKIPS Call

The SKIPS subroutine is valid only for an SRM-structured file. It positions a file to a section boundary by skipping forward or backward. The direction of the skip is determined by the sign of the count parameter.

SKIPS format is shown in figure 6-25.

When the sc parameter is 0, the file is skipped to the next end-of-section control word when the current position of the file is not at end-of-section. No action is taken if the file is already positioned at a section boundary.

When sc is a positive nonzero value, the file is skipped forward sc full sections. If the file is positioned at some point within a record when the call is made, the file is moved forward to the next end-of-section control word and then skipped forward sc sections.

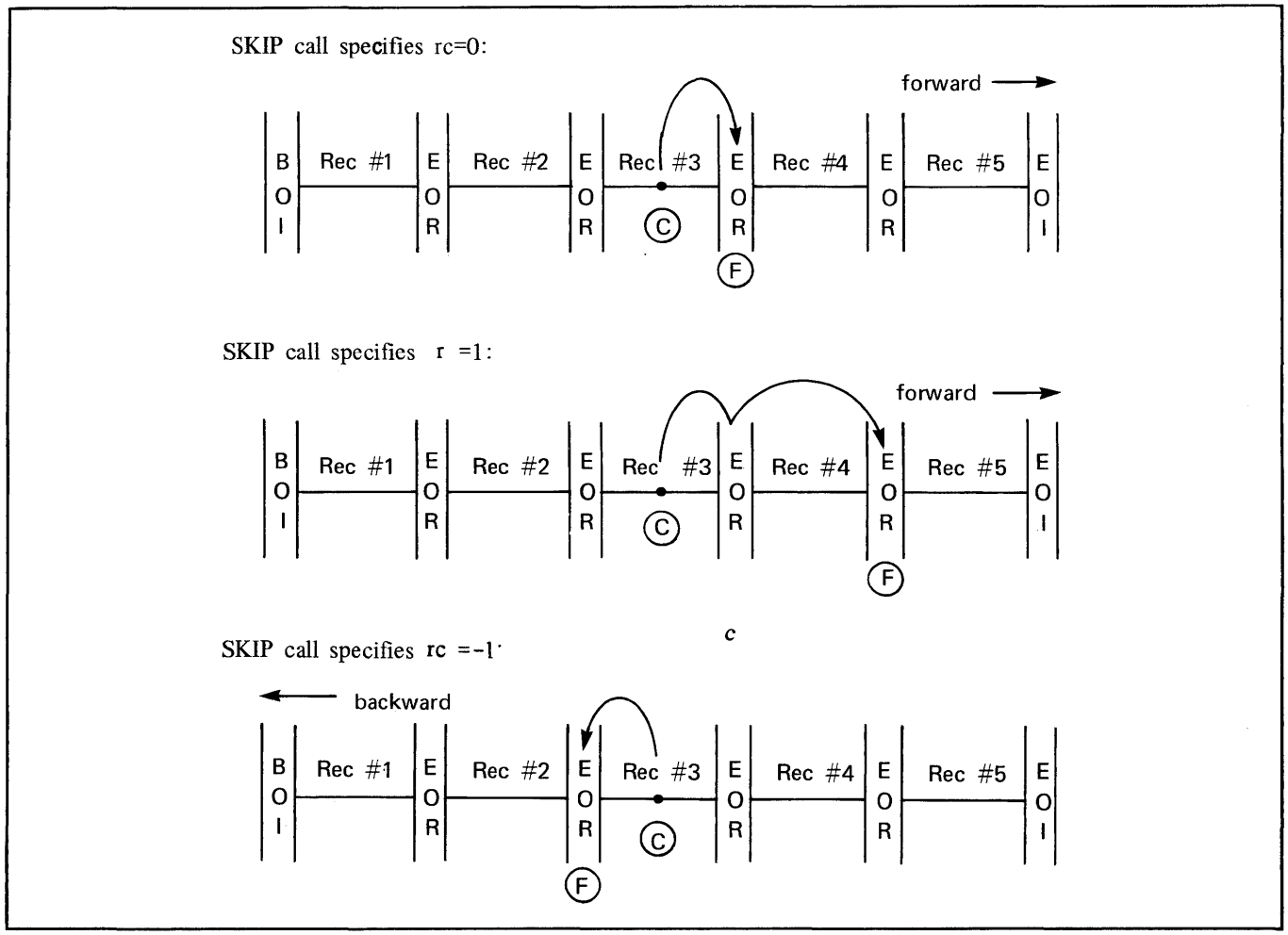


Figure 6-26. Example of SKIP Positioning

When sc is a negative value, the file is skipped backward. If the file is positioned somewhere within the record when the call is made, the file is moved backward to the first end-of-section control word encountered, counting 1 for this action. If the file is at a section boundary, the file is skipped backward the number of full sections specified by the negative sc.

When the file is positioned within a record, SKIPS action is the same as shown in figure 6-26.

SETFIT and GETFIT Calls

These subroutines access a specified field of the FIT.

GETFIT returns the value of the field to the program.

SETFIT sets a field to the value specified in the program.

FIT fields are identified by keyword mnemonics listed in table 6-3. Only the fields marked in the appropriate column of the table can be accessed by these calls.

GETFIT and SETFIT formats are shown in figure 6-27.

```
CALL Q7SETFIT (ifn, keyword, value)

CALL Q7GETFIT (ifn, keyword, value)

ifn       Internal file name of FIT

keyword   FIT field mnemonic

value     Value to be retrieved from or set into the
          FIT field
```

Figure 6-27. SETFIT and GETFIT Formats

STATUS Call

The STATUS subroutine retrieves the system error response for the last system message issued. It also can be used to relinquish control of the central processing unit until input/output is complete.

STATUS format is shown in figure 6-28.

TABLE 6-3. SYSTEM RECORD MANAGER FIT FIELDS

Keyword	Description	Can Be Referenced		
		GENFIT GENFITX	SETFIT	GETFIT
ACS	File access.	X	X	X
ALPHA1	Contents of Alpha(1).			X
ALPHA2	Contents of Alpha(2).			X
BETA1 thru BETA5	Contents of Beta(1) through Beta(5).			X
BADD	Starting virtual address of buffer where data transfer requests will deposit or obtain information.			
BLEN	Length of virtual range, in small pages, of this buffer.			
BDRY	Boundary condition.		X	X
BUF1	Buffer 1 descriptor word (refers to entire word 6 containing blen and badd).		X	X
BUF2	Buffer 2 descriptor word.		X	X
BVA	Base virtual address for mapping in file.	X	X	X
CI	Current index.			X
CLAS	Security access code.	X	X	X
DMODE	Data mode. Declares type of data occurring in a record for SRM-structured files: 0 = Binary 1 = ASCII		X	X
DRB	Directory buffer address.		X	X
EDP	End-of-data pointer.		X	X
EDX	End-of-data exit address.		X	X
EEA	Error exit address.	X	X	X
FA	File attributes (refers to entire word 1 containing the eight keyword fields).	X	X	X
FLEN	File length.		X	X
FO	File organization.	X	X	X
FP	File position.		X	X
FS	File status.			
IL	Index limit.		X	X
IOC	Input/output connector number #0 to #F, #12 to #47.	X	X	X
LFN	Logical file name in ASCII.		X	X
LOP	Last operation.		X	X
MCAT	Management category for this file.	X	X	X

TABLE 6-3. SYSTEM RECORD MANAGER FIT FIELDS (Contd)

Keyword	Description	Can Be Referenced		
		GENFIT GENFITX	SETFIT	GETFIT
MODE	Mode in which file is to be used.	X	X	X
OCS	Open/close.		X	X
PRP	Preceding record pointer.			
PTL	Partial transfer length.		X	X
RL	Record length.			
SBC	Segment byte count.			
TPM	Tape mode.		X	X
TYPE	Type of device or file.	X	X	X
UBUF	Byte count for output tape blocks.		X	X
UNIT	Logical unit on which this file resides.	X	X	X

CALL Q7STATUS (ifn, loc, cpu, err)

ifn Internal file name of the FIT

loc Variable to which the system returns status in the format shown below

cpu Indicator that control is to be relinquished when input/output is in process. Optional.

- 0 Give up control of the CPU as long as input/output for task is in process. Default.
- 1 Give up control of the CPU if input/output for this file is in process.
- 2 Do not give up control of the CPU.

err Variable to which system returns error code listed in table B-2.

Format of the status word returned is:

Bits	Length in Bits	Field
0	1	Busy bit. Set if input/output in process.
1-3	3	Unused
4-15	12	Function code with meaning indicated in table 6-2.
16-23	8	Central system error code. See the cerr field of the system message associated with the function for an explanation of the code.
24-47	24	Peripheral system error code. See the serr field of the system message associated with the function for an explanation of the code.
48-63	16	Response code

Figure 6-28. STATUS Format

Error information is returned to the program at two levels. The code returned to the variable established by the fourth parameter in the call is a generalized error code. It defines the type of error but not necessarily the system request that caused the error.

Detailed error information can be obtained through the second parameter in the call. Interpretation of these codes can be found at the system message description for various functions. If SRM detects an error condition without issuing a system message (such as an illegal parameter in processing an implicit file), the response code field of the status word is set to #FFFF and the function code set as listed in table 6-2.

TERM Call

The TERM subroutine terminates task execution. It is not required, since the program can be ended normally through the FORTRAN STOP statement.

TERM must be used to pass a return code back to the controller of the executing program. (Return codes are tested by a TV control statement in a batch job.) It is also used when the drop file for the executing task is to be preserved for future restart.

TERM format is shown in figure 6-29. Only the first parameter is required.

TPFCN Call

The TPFCN subroutine performs explicit tape functions available through the operating system. The functions that can be executed are shown with the format of TPFCN in figure 6-30.

The STATUS subroutine is required after TPFCN.

CALL Q7TERM (msg, rest, c, rc)

msg	Two-word array the subroutine requires for storing Alpha words of system message
rest	Virtual bit address at which the program is to be resumed when it is restarted
c	Drop file disposition code: 0 Preserve for task restart; default 1 Destroy 2 Preserve and abort
rc	Return code 0 through 255 to be passed back to the task controller.

Figure 6-29. TERM Format

WEOx Calls

These three subroutines are valid only for an SRM-structured file being written. They cause a control word to be written with these indicators:

WEOR	Write end-of-record
WEOS	Write end-of-section
WEOI	Write end-of-information

Formats for these subroutines are shown in figure 6-31. See figure 3-4 for a description of control words.

Any output file being processed by implicit input/output should have a call to WEOI before file close.

CALL Q7TPFCN (ifn, op, fadd)

ifn Internal file name of the FIT

op Keywood indicating tape function to be performed, as noted below.

Must be constant shown left-justified with blank fill.

fadd Value for appropriate keyword, as noted below. If this parameter is not applicable to the keyword specified, it should be set to 0.

op Parameter	Function	Related fadd Parameter
REWIND	Rewind volume	Not applicable
UNLOAD	Unload volume	Not applicable
WEOF	Write end-of-file	Not applicable
SKIPR	Space forward fadd records	Number of records to skip
REOF	Read to end-of-file	Not applicable
BKSPR	Backspace fadd records	Number of records to backspace
BKSPF	Backspace fadd files	Number of files to backspace
DENSITY	Set density	Density: 0 200 bpi 1 556 bpi 2 800 bpi 3 1600 bpi
SEEK	Seek	Not applicable
ERASE	Perform fadd erasures of 6 inches each	Number of erasures
STATUS	Return status to fadd	Variable to receive status code in same format as STATUS call parameter

Figure 6-30. TPFCN Format

WRITE Call

The WRITE subroutine is valid only for a file being accessed by explicit input/output. It transfers data from a buffer to a file in SRM-structured format or to a file in unstructured format. Parameters of the call determine the buffer from which data is to be written and whether or not the central processing unit is to be relinquished while the data transfer is taking place.

WRITE format is shown in figure 6-24.

CALL Q7WEOR (ifn)

CALL Q7WEOS (ifn)

CALL Q7WEOF (ifn)

ifn Internal name of FIT

Figure 6-31. WEOx Formats

This section describes the System Request Language feature, which is designed to act as an interface between user programs and the CYBER 200 Operating System.

OVERVIEW

The System Request Language (SRL) is a set of user-callable subroutines which provides an interface between user programs written in FORTRAN or Assembler (META) languages and the CYBER 200 Operating System. SRL adds an additional level of subroutines of the format shown in figure 7-1 to any program which uses SRL. The SRL subroutines are in the public file SYSLIB.

```
Q5xxxxxx(p1,p2,p3, . . . , pn)

xxxxxx    Mnemonic description of the function
          to be performed. The calling sequence
          is the standard FORTRAN calling sequence.
```

Figure 7-1. Format of System Request Language Call

No Assembler procedures are supplied by SRL. Assembler language users must match the FORTRAN calling sequence.

In the descriptions of the SRL subroutines which follow, apostrophes have been used to denote literals. The apostrophes are not needed if the literal string is defined with a Hollerith statement.

In the SRL subroutines, parameters can be specified as stand-alone keywords or as keyword-parameter pairs. Stand-alone keywords are variables or constants containing the ASCII characters of the keyword, for example, 'SAVE'. These parameters are used to indicate the selection of certain options that are available on certain system messages.

Table 7-1 gives the correspondence between the various SRL subroutines and the appropriate system message. For a further explanation of system messages, refer to the CYBER 200 Operating System Reference Manual, Volume 2 of 2.

TABLE 7-1. SYSTEM REQUEST LANGUAGE SUBROUTINES-TO-FUNCTION CORRESPONDENCE

SRL Routine	Function	System Message
Q5DCDPFI	Decode a file index entry.	(No message)
Q5GETACT	Get user accounting information.	USER/ACCOUNTING COMMUNICATION
Q5GETMCE	Get message from task's controllee.	GET MESSAGE FROM CONTROLLEE
Q5GETMCR	Get message from task's controller.	GET MESSAGE FROM CONTROLLER
Q5GETMOP	Get message from the operator.	GET MESSAGE FROM OPERATOR
Q5GETTN	Get task information.	MISCELLANEOUS
Q5INIT	Initialize controllee.	INITIALIZE CONTROLLEE
Q5LFIPOL	Get file index entries for pool files.	LIST FILE INDEX
Q5LFIPRI	Get file index entries for private files.	LIST FILE INDEX
Q5LFIPUB	Get file index entries for public files.	LIST FILE INDEX
Q5SNDMCE	Send message to the task's controllee.	SEND MESSAGE TO CONTROLLEE
Q5SNDMCR	Send message to the task's controller.	SEND MESSAGE TO CONTROLLER
Q5SNDMJC	Send message to the task's level-1 controller (job controller).	SEND MESSAGE TO CONTROLLER

TABLE 7-1. SYSTEM REQUEST LANGUAGE SUBROUTINES-TO-FUNCTION CORRESPONDENCE (Contd)

SRL Routine	Function	System Message
Q5TERM	Terminate task.	TERMINATE
Q5TERMCE	Disconnect controllee.	DISCONNECT CONTROLLEE
Q5TIME	Get the system time and date.	MISCELLANEOUS

Keyword-parameter pairs are two consecutive parameters. The first is similar to the stand-alone keyword; a literal which indicates the significance of the second parameter of the pair. The second parameter is a variable, constant, or array used to pass information to or from SRL. For example, 'DATE=',ASCIDATE; where 'DATE=' is the literal, and ASCIDATE is the variable used to receive the ASCII-formatted date.

In the descriptions of type parameters for each subroutine call, the two types of parameters can be distinguished by the equal sign (=), which is the last character of the pair type keyword.

Keywords are mnemonics and descriptive of their significance. Generally, the keywords are the same as the corresponding field descriptors shown in table 7-1 for system messages. An example of a complete calling sequence is shown in figure 7-2.

The call in figure 7-2 will send the message to the controller. The message is 'READING TAPE' and is 12 bytes in length. The request status will be returned in a variable called STATUS.

```
CALL Q5SNDMCR('MSG=','READING TAPE','LEN=',
     12,'STATUS=',STATUS)
```

Figure 7-2. System Request Language Subroutine Calling Sequence

NO-OP KEYWORDS

Two special no-operation keywords are designed to facilitate selection of optional parameters. These keywords are available with all SRL subroutines and can be specified more than once.

'NOP' is a stand-alone keyword and 'NOP=' is a pair type keyword (left side only). They are used in calls to subroutines such as Q5SNDMCR where the 'REJECT' option is to be used in some instances, but not in others. Thus the calling sequence would include a variable containing the keyword 'REJECT' in one instance, and the keyword 'NOP' in another instance.

For example, a user can include code to terminate a task abnormally, resulting in an error being issued, or normally, with no error issued. The code to do this would be as follows:

```
EC = 'NOP'
IF (ERROR) EC = 'ABORT'
CALL Q5TERM (EC)
```

Most parameters for SRL subroutines are optional. In the descriptions which follow, default values are given. Required parameters for each subroutine are so specified. ASCII fields, in the descriptions given, are right-justified with zero fill.

STATUS CODES

System Request Language uses returned status codes to indicate errors. A caller of an SRL subroutine must specify the 'STATUS=' keyword-parameter pair to obtain the returned status codes. If a fatal error occurs, and 'STATUS=' was not specified, the task is terminated with a return code of 8. The error message is then sent to the dayfile (batch job) or to the user terminal (interactive job). Control is then returned to the next higher level controllee.

Keyword and byte (8-bit bytes) type parameters must begin on byte boundaries unless otherwise stated in the descriptions.

Q5DCDPFI

Q5DCDPFI decodes a formatted PFI contained in the SRL-defined buffer into separate areas. Q5LFIPRI, Q5LFIPUB, or Q5LFIPOL must be called prior to calling this routine. If they are not, an error is returned. Any number of fields can be obtained by the use of keyword-parameter pairs. Figure 7-3 shows the format of the Q5DCDPFI subroutine call.

Q5GETACT

The Q5GETACT subroutine is used to obtain user accounting information through the USER/ACCOUNTING COMMUNICATION system message. The format of the Q5GETACT subroutine is shown in figure 7-4.

Q5GETMCE

The Q5GETMCE subroutine is used to obtain a message from the task's controllee using the GET MESSAGE FROM CONTROLLEE system message. Options are selected through the use of stand-alone keywords and keyword-parameter pairs. The format of the Q5GETMCE subroutine call is shown in figure 7-5.

Calling Parameters

'ENTRY=' Relative entry number of desired file index entry (beginning with 1). Default - the last value of 'ENTRY=' plus 1 (initial call after use of Q5LFIPRI, Q5LFIPUB, or Q5LFIPOL 'ENTRY=' is 1).

Return Parameters

'STATUS=' Request status. Possible values:

0 thru 100
104, 105

'ERRMSG=' Byte array to which SRL returns a formatted error message.

'ERRLEN=' Length of the formatted error message.

'FILENAM=' ASCII name of the file, left-justified with blank fill. The paired parameter must begin on a word boundary.

'MCAT=' File management category. Possible values:

0	Mass storage file
1	Scratch file
2	Output file
3	Write temporary file
5	User-created drop file
6	System-created drop file
7	Batch file

'SADDR=' Disk address of the first page of the file.

'UNIT=' Logical unit number of the disk on which the file resides.

'WLEN=' File length.

'DUP=' Duplicate file name flag. It is set to #D if a duplicate file entry exists; otherwise, it is set to 1.

'GIVETU1=' Binary user number of the user who gave this file to the user 1 routine.

'TYPE=' File type. Possible values:

0	Physical data
1	Virtual data
2	Virtual code

'SLEV=' Security level. Possible values:

0 thru 255

'ACS=' File access permission. Possible values:

0	No access is permitted (same as acs = 4)
1	Write access is permitted (same as acs = 5)
2	Read access is permitted (same as acs = 6)
3	Read and write access is permitted (same as acs = 7)

'TORG=' File creation time. The integer representation of system clock time, in seconds, since midnight, when this file was originated.

'TLR=' Time of last open request. The integer representation of system clock time, in seconds, since midnight, when the file was last opened.

'DC=' Disposition code. Possible values:

SC	Scratch (destroyed at end of task); default.
PR	Print on any available printer

Figure 7-3. Q5DCDPFI Subroutine (Sheet 1 of 3)

PU	Punch	
IN	Input for batch processing	
PF	Permanent file	
P1	501 line printer only	Reserved for CYBER 200 Link Station
P2	512 line printer only	
LR	580-12 printer only	
LS	580-16 printer only	
LT	580-20 printer only	

Codes IN and PF are appropriate for disposition through an Access Station only.

'IC=' Internal characteristics indicating the format of the file. Possible values:

AS	Format is 8-bit ASCII. If the file has a disposition code of PR, the file has ANSI carriage control characters.
BI	Binary format.
PA or Binary zero	Format is 8-bit ASCII. If the file has a disposition code of PR, the file has ASCII carriage control characters.

'EC=' External characteristics indicating the print or punch representation of the file. Possible values:

26	Punch in 026 keypunch format	
29 or Binary 200	Punch in 029 keypunch format	
*B	Punch in CYBER 200 binary format	
80	Punch in 80 column binary format	
B4	BCD 48-character print train	Reserved for CYBER 200 Link Station
B6	BCD 64-character print train	
A4	ASCII 48-character print train	
A6	ASCII 64-character print train	
A9	ASCII 95-character print train	

'CM=' Conversion mode indicating the type of conversion to be done on the Access Station (st=AST). Possible values:

DI	Display code (64-character set)
EC	Extended display code (128-character set)
BI	Binary

'ST=' 3-character site identifier. Possible values:

AST	Access Station
ADA	Reserved for the CYBER front-end portion of the CYBER 200 Link Station
ADZ or ADX	Reserved for the CYBER 200 portion of the CYBER 200 Link Station

'OT=' Origin type of the file destined for the Access Station (st=AST). Possible values:

B	Batch
E	Remote batch
I	Interactive

'REP=' Retention period of the file in days (integer).

'ZIP=' Site identifier. See site analyst.

'TID=' Terminal identifier. For a file destined for the Access Station, a user number of 1 to 7 characters that identifies the user on processor ST to which the file is routed. The central site is indicated by TID=0. (Not meaningful for files destined for the CYBER 200.) (ASCII.)

'DI=' Delivery information for the Access Station. One to eight 8-bit characters held with the file.

Figure 7-3. Q5DCDPFI Subroutine (Sheet 2 of 3)

'DORG='

Date this file was originated, in the format:

yy	Last two digits of the year
ddd	Number of days in Julian date format since the beginning of the year, 1 through 366

'DOLA='

Date of the last access to this file. Of the same format as the 'DORG=' field.

'DOLM='

Date of the last open request to this file with write access. Of the same format as the 'DORG=' field.

'LOCAL='

Indicates whether the file is local or permanent. Possible values:

0	Permanent
1	Local

'CONT='

File contiguity, which is set at file creation time. This field is zero if the file need not be created contiguously (that is, can be in two segments), or is set to 1 if the file must be created contiguously.

'EXT='

Extension permission flag, which is set at file creation or open. This field is zero if extensions are allowed and set to 1 if extensions are not allowed.

'TOLM='

Time, in seconds, since midnight, when this file was last opened with write access.

'NAC='

Access Station area code.

'SHACC='

Shared access permission for pool files. Possible values:

0	No shared access is permitted (same as shacc = 4)
1	Write shared access is permitted (same as shacc = 5)
2	Read shared access is permitted (same as shacc = 6)
3	Read and write shared access is permitted (same as shacc = 7)

'LODLEN='

Length, in small pages, of the program's drop file. Meaningful only for type 2, otherwise this field is zero.

'FACT='

ASCII account number, left-justified with blank fill. The paired parameter must begin on a word boundary.

'ATSUF='

Attached suffix. Possible values:

0	File is not attached
=0	File is attached

ATSUF contains a value relating to the attached suffix (ASCII user number). Possible values:

1	Suffix A
2	Suffix B
3	Suffix C
4	Suffix D

'AUSER='

ASCII user number.

'POOLNAM='

Pool name, in ASCII, left-justified, with blank fill. The paired parameter must begin on a word boundary.

'ORIGLEN='

Original file length, in small pages, requested during file creation.

'SEGLEN$_i$='

Length of the i^{th} segment of the file in small pages; i has a value of 1 to 4.

'SEGADR$_i$='

Disk sector address of the i^{th} segment of the file; i has a value of 1 to 4.

Figure 7-3. Q5DCDPFI Subroutine (Sheet 3 of 3)

Calling Parameters

None.

Return Parameters

'STATUS='	Request status. Possible values: 0 thru 100 200 thru 202
'ERRMSG='	Byte array to which SRL returns a formatted error message.
'ERRLEN='	Length of the formatted error message.
'CPUTIME='	User execution CPU time, in microseconds.
'MEMUSE='	Memory usage. At the end of each account period, (current working set size) user CPU time for current accounting period is computed and added to a running total kept in this field.
'MTXFER='	Number of 16-bit bytes transferred to and/or from magnetic tape.
'MTACCES='	Number of magnetic tape reads and writes.
'MTNONIO='	Number of non-I/O magnetic tape operations.
'SYSTIME='	System CPU execution time, in microseconds.
'DPLEPX='	Number of large page explicit reads and writes to or from mass storage.
'DPLIMP='	Number of large page implicit writes to mass storage.
'DPSEXP='	Number of small page explicit reads and writes to and from mass storage.
'DPSIMP='	Number of small page implicit writes to mass storage.
'DPXEXP='	Number of sectors transferred to mass storage for explicit reads and writes.
'DPXFIMP='	Number of sectors transferred to mass storage for implicit writes.
'DPLPFLT='	Number of mass storage reads due to large page faults.
'DPSPFLT='	Number of mass storage reads due to small page faults.
'CWSSIZ='	Current working size.
'VSCALL='	Number of virtual system user calls made.
'SYSCHRG='	Number of 16-bit bytes transferred to or from tape files.

Figure 7-4. Q5GETACT Subroutine

Calling Parameters

'LEN='	Length of the message buffer in bytes. Maximum is 4096 bytes; default is 80 bytes.
'STD'	Indicates use of standard delimiters (period, blank, comma, slash, equal, plus, minus, left parenthesis, and right parenthesis). Default - no editing of message.
'SYD'	Indicates use of system delimiters (defined as an installation parameter). Default - no editing of message.
'NULLFILL'	Indicates fill character of binary zero (if 'STD' or 'SYD' is specified). Default - blank fill.
'RJUSTIFY'	Indicates right justification of symbols (if 'STD' or 'SYD' is specified). Default - left-justification.
'SAVE'	Indicates that the system buffer space is to be saved. Default - release the buffer space.

Return Parameters

'STATUS='	Request status. Possible values: 0 thru 100 200 thru 202 340 thru 342 344 345
'ERRMSG='	Byte array to which SRL returns a formatted error message.
'ERRLEN='	Length of formatted error message.
'MSG='	Received (and optionally edited) message. Required parameter.
'MSGLEN='	Number of bytes received.
'DB='	Descriptor block number of the controllee that sent the message.
'LEVEL='	Level of the controllee that sent the message.

Note: The parameters 'STC' and 'SYD' are mutually exclusive, and only one can be specified. If both are specified an error is returned.

Figure 7-5. Q5GETMCE Subroutine

Q5GETMCR

The Q5GETMCR subroutine is used to obtain a message from the task's controller using the GET MESSAGE FROM CONTROLLER system message. Options are selected through the use of stand-alone keywords and keyword-parameter pairs. The format of the Q5GETMCR subroutine is shown in figure 7-6.

Calling Parameters

'LEN='	Length of the message buffer in bytes. Maximum is 4096 bytes; default is 80 bytes.
'STD'	Indicates use of standard delimiters (period, blank, comma, slash, equal, plus, minus, left parenthesis, and right parenthesis). Default - no editing of message.
'SYD'	Indicates use of system delimiters (defined as an installation parameter). Default - no editing of message.
'NULLFILL'	Indicates fill character of binary zero (if 'STD' or 'SYD' are specified). Default - blank fill.
'RJUSTIFY'	Indicates right-justification of symbols (if 'STD' or 'SYD' are specified). Default - left-justification.
'REJECT'	Indicates if a message is not present; returns an error code. Default - wait until a message is available.
'SAVE'	Indicates that the system buffer space is to be saved. Default - release the buffer space.

Return Parameters

'STATUS='	Request status. Possible values: 0 thru 100 200 thru 202 340 thru 343
'ERRMSG='	Byte array to which SRL returns a formatted error message.
'ERRLEN='	Length of formatted error message.
'MSG='	Received (and optionally edited) message. Required parameter.
'MSGLEN='	Number of bytes received.
'DB='	Descriptor block number of the controller that sent the message.
'LEVEL='	Level of the controller that sent the message.

Note: The parameters 'STD' and 'SYD' are mutually exclusive, and only one can be specified. If both are specified, an error is returned.

Figure 7-6. Q5GETMCR Subroutine

Q5GETMOP

The Q5GETMOP subroutine obtains a message from the operator using the GET MESSAGE FROM OPERATOR system message. As with other SRL subroutines, options are specified through the use of stand-alone keywords and keyword parameter pairs. The format of the Q5GETMOP subroutine is shown in figure 7-7.

Calling Parameters

'LEN='	Length of the message in bytes. Maximum is 4096 bytes; default is 80 bytes.
'STD'	Indicates use of standard delimiters (period, blank, comma, slash, equal, plus, minus, left parenthesis, and right parenthesis). Default - no editing of message.
'SYD'	Indicates use of system delimiters (defined as an installation parameter). Default - no editing of message.
'NULLFILL'	Indicates fill character of binary zero (if 'STD' of 'SYD' are specified). Default - blank fill.
'RJUSTIFY'	Indicates right-justification of symbols (if 'STD' or 'SYD' are specified). Default - left-justification.
'REJECT'	Indicates if a message is not present; returns an error code. Default - wait until a message is available.

Return Parameters

'STATUS'	Request status. Possible values: 0 thru 100 200 thru 202 340 thru 342
'ERRMSG='	Byte array to which SRL returns a formatted error message.
'ERRLEN='	Length of the formatted error message.
'MSG='	Received (and optionally edited) message. Required parameter.
'MSGLEN='	Number of bytes received.
'DB='	Descriptor block of the operator task.

Note: The parameters 'STD' and 'SYD' are mutually exclusive and only one can be specified. If both are specified, an error is returned.

Figure 7-7. Q5GETMOP Subroutine

Q5INIT

The Q5INIT subroutine initializes a controllee using the INITIALIZE CONTROLLEE system message. Stand-alone keywords and keyword-parameter pairs are used to select desired options. The format of the Q5INIT subroutine is shown in figure 7-8.

Calling Parameters

'FILENAM='	Name of the controllee file to be initialized. Required parameter. Default - none.
'WAIT'	Indicator that this controllee is to be started running and this task stopped as soon as the controllee is initialized. Default - restart this task after initializing controllee.
'TLIMIT='	Time limit for the controllee program in microseconds. Default - task's time limit is used.

Return Parameters

'STATUS='	Request status. Possible values: 0 thru 100 200 thru 202 350 thru 367
'ERRMSG='	Byte array to which SRL returns a formatted error message.
'ERRLEN='	Length of the formatted error message.
'DB='	Controllee program's descriptor block number. If the controllee program is disconnected, and then reconnected, this number could change.

Figure 7-8. Q5INIT Subroutine

Q5GETTN

The Q5GETTN subroutine obtains a task's binary and drop file name plus miscellaneous information through the use of the MISCELLANEOUS system message. The format of this subroutine is shown in figure 7-9.

Calling Parameters

None.

Return Parameters

'STATUS='	Request status. Possible values: 0 thru 100 200 thru 202
'ERRMSG='	Byte array to which SRL returns a formatted error message.
'ERRLEN='	Length of the formatted error message.
'BINARY='	Source file name for this task, in ASCII, right-justified with blank fill.
'DROPFIL='	Drop file name for this task, in ASCII, right-justified with blank fill.
'SUFFIX='	Suffix for this task (ASCII character A, B, C, or D).
'LEVEL='	Level of this task in the controllee chain.
'USERID='	ASCII user number, right-justified with blank fill.
'PRIV='	Privileged user flag. Possible values: =0 Nonprivileged ≠0 Privileged

Note: If none of the parameters 'BINARY=' through 'PRIV=' are specified, an error is returned.

Figure 7-9. Q5GETTN Subroutine

Q5FIPOL

The Q5LFIPOL subroutine obtains the formatted file index entries for a set of pool files by using the LIST FILE INDEX system message. The indices are maintained in an SRL defined buffer and are referenced by the decode routine Q5DCDPFI. Options are selected through the use of stand-alone keywords and keyword-parameter pairs. The format of the Q5LFIPOL subroutine is shown in figure 7-10.

Q5FIPRI

The Q5LFIPRI subroutine obtains the formatted file index entries for all private files that qualify, through the use of the LIST FILE INDEX system message. The indices are maintained in an SRL-defined buffer and are referenced by the decode subroutine Q5DCDPFI. Options are selected through the use of stand-alone keywords and keyword-parameter pairs. The format of the Q5LFIPRI subroutine is shown in figure 7-11.

Q5FIPUB

Q5LFIPUB obtains file index table entries for all public files that qualify, through the use of the LIST FILE INDEX system message. The indices are maintained in an SRL-defined buffer and are referenced by the decode routine Q5DCDPFI. Options are selected through the use of stand-alone keywords and keyword-parameter pairs. The format of the Q5LFIPUB call is shown in figure 7-12.

Calling Parameters

'POOLNAM=' Name of the pool containing the files for which file index entries are to be obtained. Default - none. Required parameter.

'FILENAM=' An array of the file names for which PFIs are to be obtained (ASCII, left-justified with blank fill). Default - all file names belonging to this pool.

'FNCOUNT=' The number of files in the 'FILENAM=' array. Default - 1 (0 if 'FILENAM=' is not specified).

'STRING' Indicator that the elements specified by 'FILENAM=' are to be used as character strings. Q5LFIPOL will return file index entries for all files in the specified pool whose names begin with one of the strings. Default - no string matching.

'UNIT=' Logical unit number containing files for which file index entries are to be obtained. Default - all units defined in the system.

'DC=' Disposition code. File index entries are returned only for files with the selected DC. Possible values:

SC	Scratch (destroyed at end of task); default	
PR	Print on any available line printer	
PU	Punch	
IN	Input for batch processing	
PF	Permanent file	
P1	501 line printer only	Reserved for CYBER 200 Link Station
P2	512 line printer only	
LR	580-12 line printer only	
LS	580-16 line printer only	
LT	580-20 line printer only	

Codes IN and PF are appropriate for disposition through an Access Station only.

'IC=' Internal characteristics. File index entries are returned only for files with the selected IC. Possible values:

AS	Format is 8-bit ASCII. If the file has a disposition code of PR, the file has ANSI carriage control characters.
BI	Format is binary.
PA or Binary zero	Format is 8-bit ASCII. If the file has a disposition code of PR, the file has ASCII carriage control characters. Default.

'EC=' External characteristics. File index entries are returned only for files with the selected EC. Possible values:

26	Punch in 026 keypunch format	
29 or Binary zero	Punch in 029 keypunch format; default	
*B	Punch in CYBER 200 format	
80	Punch in 80-column binary format	
B4	BCD 48-character print train	Reserved for CYBER 200 Link Station
B6	BCD 64-character print train	
A4	ASCII 48-character print train	
A6	ASCII 64-character print train	
A9	ASCII 95-character print train	

'ST=' A 3-character site identifier. File index entries are returned only for files with the select ST. Possible values:

AST	Access Station
ADA	Reserved for the CYBER front-end portion of the CYBER 200 Link Station
ADZ or ADX	Reserved for the CYBER 200 portion of the CYBER 200 Link Station

'ZIP=' The zip code for the site identifier specified in the 'ST=' field. See site analyst.

Figure 7-10. Q5LFIPOL Subroutine (Sheet 1 of 2)

Return Parameters

'STATUS='	Request status. Possible values: 0 thru 100 200 thru 202 300, 301
'ERRMSG='	Byte array to which SRL returns a formatted error message.
'ERRLEN='	Length of the formatted error message.
'NFILES='	Number of file index entries returned in the SRL-defined buffer.

Figure 7-10. Q5LFIPOL Subroutine (Sheet 2 of 2)

Calling Parameters

'FILENAM='	An array of the file names for which PFIs are to be obtained (ASCII right-justified with blank fill). Default - all private file names belonging to this user.
'FNCOUNT='	The number of files in the 'FILENAM=' array. Default - 1 (0 if 'FILENAME=' is not specified).
'STRING'	Indicator that the elements of the array specified by 'FILENAM=' are to be used as character strings. Q5LFIPRI will return file index entries for all private files owned by this user, whose names begin with one of the strings. Default - no string matching.
'UNIT='	Logical unit number containing the files for which file index entries are to be obtained. Default - all units defined to the system.
'DC='	Disposition code. File index entries are returned only for files with the selected DC. Possible values: SC - Scratch (destroyed at end of task); default PR - Print on any available line printer PU - Punch IN - Input for batch processing PF - Permanent file P1 - 501 printer only P2 - 512 printer only LR - 580-12 printer only LS - 580-16 printer only LT - 580-20 printer only (P1, P2, LR, LS, LT: Reserved for CYBER 200 Link Station) Codes IN and PF are appropriate for disposition through an Access Station only.
'IC='	Internal characteristics. File index entries are returned only for files with the selected IC. Possible values: AS - Format in 8-bit ASCII. If the file has a disposition code of PR, the file has ANSI carriage control characters. BI - Format in binary. PA or Binary zero - Format is 8-bit ASCII. If the file has a disposition code of PR, the file has ASCII carriage control characters. Default.
'EC='	External characteristics. File index entries are returned only for files with the selected EC. Possible values: 26 - Punch in 026 keypunch format 29 or Binary zero - Punch in 029 keypunch format; default *B - Punch in CYBER 200 format 80 - Punch in 80-column binary format B4 - BCD 48-character print train B6 - BCD 64-character print train A4 - ASCII 48-character print train A6 - ASCII 64-character print train A9 - ASCII 95-character print train (B4, B6, A4, A6, A9: Reserved for CYBER 200 Link Station)

Figure 7-11. Q5LFIPRI Subroutine (Sheet 1 of 2)

'ST='

A 3-character site identifier. File index entries are returned only for files with the selected ST. Possible values:

AST	Access Station
ADA	Reserved for the CYBER front-end portion of the CYBER 200 Link Station
ADZ or ADX	Reserved for the CYBER 200 portion of the CYBER 200 Link Station

'ZIP='

The zip code for the site identifier specified in the 'ST=' field. For possible values for this field, contact a site analyst.

Return Parameters

'STATUS='

Request status. Possible values:

0 thru 100
200 thru 202
300, 301

'ERRMSG='

Byte array to which SRL returns a formatted error message.

'ERRLEN='

Length of error message.

'NFILES='

Number of file index entries returned in SRL-defined buffer.

Figure 7-11. Q5LFIPRI Subroutine (Sheet 2 of 2)

Calling Parameters

'FILENAM='

An array of the filenames for which file index entries are to be obtained in ASCII, right-justified with blank fill. Default - all public file names.

'FNCOUNT='

The number of files in the 'FILENAM=' array. Default - 1 (0 if 'FILENAM=' is not specified).

'STRING='

Indicator that the elements of the array specified by 'FILENAM=' are to be used as strings. Q5LFIPUB will return file index entries for all public files whose names begin with one of the strings. Default - no string matching performed.

'UNIT='

Logical unit number containing files for which file index entries are to be obtained. Default - all units defined to the system.

'DC='

Disposition code. File index entries are returned only for files with the selected DC. Possible values:

SC	Scratch (destroyed at end-of-task); default	
PR	Print on any available line printer	
PU	Punch	
IN	Input for batch processing	
PF	Permanent file	
P1	501 printer only	Reserved for CYBER 200 Link Station
P2	512 printer only	
LR	580-12 printer only	
LS	580-16 printer only	
LT	580-20 printer only	

Codes IN and PF are appropriate for disposition through an Access Station only.

'IC='

Internal characteristics. File index entries are returned only for files with the selected IC. Possible values:

AS	Format in 8-bit ASCII. If the file has a disposition code of PR, the file has ANSI carriage control characters.
BI	Format in binary.
PA or Binary zero	Format is 8-bit ASCII. If the file has a disposition code of PR, the file has ASCII carriage control characters. Default.

Figure 7-12. Q5LFIPUB Subroutine (Sheet 1 of 2)

'EC='

External characteristics. File index entries are returned only for files with the selected EC. Possible values:

26	Punch in 026 keypunch format	
29 or Binary zero	Punch in 029 keypunch format; default	
*B	Punch in CYBER 200 format	
80	Punch in 80-column binary format	
B4	BCD 48-character print train	Reserved for CYBER 200 Link Station
B6	BCD 64-character print train	
A4	ASCII 48-character print train	
A6	ASCII 64-character print train	
A9	ASCII 95-character print train	

'ST='

A 3-character site identifier. File index entries are returned only for files with the selected ST. Possible values:

AST	Access Station
ADA	Reserved for the CYBER front-end portion of the CYBER 200 Link Station
ADZ or ADX	Reserved for the CYBER 200 portion of the CYBER 200 Link Station

'ZIP='

The zip code for the site identifier specified in the 'ST=' field. For possible values for this field, contact a site analyst.

Return Parameters

'STATUS='

Request status. Possible values:

0 thru 100
200 thru 202
300, 301

'ERRMSG='

Byte array to which SRL returns a formatted error message.

'ERRLEN='

Length of error message.

'NFILES='

Number of file index entries returned in SRL-defined buffer.

Figure 7-12. Q5LFIPUB Subroutine (Sheet 2 of 2)

Q5SNDMCE

The Q5SNDMCE sends a message to the calling task's controllee using the SEND MESSAGE TO CONTROLLEE system message. Options are selected through the use of stand-alone keywords and keyword-parameter pairs. The format of the Q5SNDMCE call is shown in figure 7-13.

Q5SNDMCR

Q5SNDMCR sends a message to the calling task's controller using the SEND MESSAGE TO SYSTEM CONTROLLER system message. Options are selected thrugh the use of stand-alone keywords and keyword-parameter pairs. The format of the Q5SNDMCR is shown in figure 7-14.

Q5SNDMJC

Q5SNDMJC sends a message to the calling task's job controller (batch processor or virtual system interactive processor) using the SEND MESSAGE TO CONTROLLER system message. Options are selected through the use of stand-alone keywords and keyword-parameter pairs. The format of the Q5SNDMJC call is shown in figure 7-15.

Calling Parameters

'MSG='

Message to be sent (maximum is 4096 bytes). Default - none, required parameter.

'LEN='

Number of bytes in the message to be sent. Default - the first character of the message is assumed to be a delimiter and the message, beginning with the second character to (but not including) the next occurrence of the delimiter, is sent. If the number of bytes is greater than 4096, it is set to 4096 and treated as no error.

'REJECT'

Indicator that if the message would replace an existing message, stop running this task until the message can be sent. Default - replace existing message with this message.

Figure 7-13. Q5SNDMCE Subroutine (Sheet 1 of 2)

'REMOVE'	Remove this program from main memory prior to sending the message. Default - do not remove program from main memory.
'DB'	Descriptor block number of controllee to receive message. Default - next lower controllee.
Return Parameters	
'STATUS='	Request status. Possible values: 0 thru 100 200 thru 202 320, 321 325
'ERRMSG='	Byte array to which SRL returns a formatted error message.
'ERRLEN='	Length of error message.

Figure 7-13. Q5SNDMCE Subroutine (Sheet 2 of 2)

Calling Parameters	
'MSG='	Message to be sent (maximum is 4096 bytes). Default - none, required parameter.
'LEN='	Number of bytes in the message to be sent. Default - the first character of the message is assumed to be a delimiter and the message, beginning with the second character to (but not including) the next occurrence of the delimiter, is sent. If the number of bytes is greater than 4096, it is set to 4096 and treated as no error.
'REJECT'	Indicator that an error code is to be returned if the message cannot be sent immediately. Default - wait until message can be sent.
'REMOVE'	Remove this program from main memory prior to sending the message. Default - do not remove program from main memory.
'DB='	Descriptor block number of controller to receive message. Default - next higher controller.
Return Parameters	
'STATUS='	Request status. Possible values: 0 thru 100 200 thru 202 320 thru 324
'ERRMSG='	Byte array to which SRL returns a formatted error message.
'ERRLEN='	Length of error message.

Figure 7-14. Q5SNDMCR Subroutine

Calling Parameters	
'MSG='	Message to be sent (maximum is 4096 bytes). Default - none, required parameter.
'LEN='	Number of bytes in the message to be sent. Default - the first character of the message is assumed to be a delimiter and the message, beginning with the second character to (but not including) the next occurrence of the delimiter, is sent. If the number of bytes is greater than 4096, it is set to 4096 and treated as no error.
'REJECT'	Indicates that if the message would replace an existing message, stop running this task until the message can be sent. Default - replace existing message with this message.
Return Parameters	
'STATUS='	Request status. Possible values: 0 thru 100 200 thru 202 320 thru 324
'ERRMSG='	Byte array to which SRR returns a formatted error message.
'ERRLEN='	Length of error message.

Figure 7-15. Q5SNDMJC Subroutine

Q5TERM

The call Q5TERM is used to terminate a task and its lower level controllees using the TERMINATE system message. Various options are selected through the use of stand-alone keywords and keyword-parameter pairs. A call to Q5TERM with no parameters specified is considered nomal termination. The format of the Q5TERM call is shown in figure 7-16.

Q5TERMCE

The Q5TERMCE call will disconnect a previously initialized controllee using the DISCONNECT CONTROLLEE system message. The format of the Q5TERMCE is shown in figure 7-17.

Q5TIME

Q5TIME obtains the system time and date information using the MISCELLANEOUS system message. Various formats can be obtained by the use of the appropriate keyword-parameter pairs. The format of the Q5TIME call is shown in figure 7-18.

Calling Parameters

'RESTART'	Indicator that the drop file, scratch files, and output files are to be preserved so that the program can be restarted (these files will be local unless defined). Default - these files are destroyed.
'ABORT'	Indicates a termination state of #3D (task aborted). Overrides 'RESTART' if both are used. Default - termination state of #3E (normal termination).
'ERROR'	Indicates nonfatal errors occurred, for example, system return code = 4. Default - see 'FATAL'.
'FATAL'	Indicates fatal errors occurred, for example, system return code = 8. Default - ('ERROR' and 'FATAL' not specified) no errors occurred, for example, system return code = 0.
'RESUME='	Virtual bit address at which program is to be resumed when it is restarted. Default - resume execution at return from call to Q5TERM.

The parameters 'ERROR' and 'FATAL' are mutually exclusive and only one can be specified. If more than one is specified an error is returned.

Return Parameters

'STATUS='	Request status. Possible values: 0 thru 99
'ERRMSG='	Byte array to which SRL returns a formatted error message.
'ERRLEN='	Length of error message.

Figure 7-16. Q5TERM Subroutine

Calling Parameters

None.

Return Parameters

'STATUS='	Request status. Possible values: 0 thru 99 200 thru 202 370
'ERRMSG='	Byte array to which SRL returns a formatted error message.
'ERRLEN='	Length of error message.

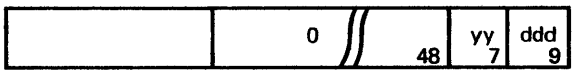
Figure 7-17. Q5TERMCE Subroutine

Calling Parameters

None.

Return Parameters

'STATUS='	Request status. Possible values: 0 thru 100 200 thru 202
'ERRMSG='	Byte array to which SRL returns a formatted error message.
'ERRLEN='	Length of error message.
'TIME'	ASCII character string in the form hh.mm.ss. The paired parameter should begin on a word boundary.
'DATE='	ASCII character string in the form mm/dd/yy. The paired parameter should begin on a word boundary.
'MASTER='	Binary value of the master clock (seconds since midnight).
'JULIAN='	Binary Julian date in the form:



where yy is the year and dd is the day within the year.

Note: If none of the parameters 'TIME=', 'DATE=', 'MASTER=' or 'JULIAN=' are specified, an error code will be returned.

Figure 7-18. Q5TIME Subroutine

ERROR PROCESSING

System Request Language uses returned status codes to indicate errors. A caller of an SRL subroutine must specify the "STATUS=" keyword-parameter pair to obtain the returned status codes. If a fatal error occurs, and "STATUS=" is not specified, the task is terminated with a return code of 8, and control is returned to the next higher level controller; the error message is sent to the dayfile (batch job) or to the user terminal (interactive job).

Also, SRL will return an error message to a user-specified byte array. The message is obtained by specification of the "ERRMSG=" keyword-parameter pair. The length of the message is obtained by specification of the "ERRLEN=" keyword-parameter pair. The maximum length of the message is 80 character bytes.

ERROR CODES

The possible values of the error codes fall into one of several ranges, depending on their significance, as shown in table 7-2.

ERROR MESSAGE FORMAT

Each error message has the format shown in figure 7-19. The System Request Language error messages are shown in table B-4.

TABLE 7-2. SYSTEM REQUEST LANGUAGE ERROR CODES

Range	Significance
0	No error.
1 thru 99	User interface error - error in keyword specification.
100 thru 199	Interface routine detected error - error in parameter passed to interface routine.
200 thru 299	Interface routine internal error - bug.
300 thru 9999	System detected error-translated by SRL to common set of error codes.
10 000 thru 11 000	Reserved for installation use.

<severity>Δ<routine>Δ<number>Δ<description>

where:

<severity>	A one-character indicator of the level of severity of the error: F=fatal or W=warning.
Δ	Blank.
<routine>	The name of the SRL routine which caused the error.
<number>	The value of the status code returned to the user.
<description>	A brief description of the error.

Figure 7-19. System Request Language Error Message Format

DEBUG, LOOK, and DUMP are utilities for testing and debugging a correctly compiled or assembled program that executes unsatisfactorily. Also available for debugging purposes is the DEBUG parameter of the LOAD control statement in section 4. These utilities can be executed either interactively or in batch mode.

Differences among these three utilities include:

DEBUG displays or alters the contents of selected locations during program execution. It is valid only with controllee files.

LOOK displays or alters the contents of selected locations in any type of file. It can be used with controllee files or data files. Its most common use is through an interactive terminal.

DUMP displays a preselected set of elements from a drop file.

Both LOOK and DEBUG use a set of directives supplied by a programmer to receive detailed control information. A batch job must have the directives on a file available to the job.

An interactive user can enter directives interactively and receive output as it is generated in response to the directive. Output from most directives is returned to the terminal; some directives can specify a file to receive output. When the utility is ready for another directive, the character ? appears at the terminal. Directives must be entered on a single line.

Typical use of the debugging utilities involves using LOOK to edit a FORTRAN source program interactively until the program compiles successfully; executing the compiled program and possibly receiving a dump on a fatal error condition, or else possibly forcing such a dump by making a DUMP request; using DEBUG to observe intermediate program values during reexecution of the program under DEBUG control; and subsequently using DEBUG or LOOK to modify the program until it executes satisfactorily.

DEBUG

Through DEBUG the user can set breakpoints in a program, then issue EXECUTE and CONTINUE directives to step through the execution of the program from one breakpoint to the next. At each breakpoint, current values of variables in the program can, for example, be dumped. DEBUG can also be used to modify, display, and dump user registers and areas in virtual memory designated by hexadecimal addresses.

A FORTRAN program being executed under DEBUG must have been compiled under the S compile option if symbolic addresses - labels, names, line numbers - are to be used in the DEBUG directives. DEBUG executes entirely within the user's virtual space, starting at hexadecimal virtual bit address #7FFF00000000 and extending upwards; therefore, the program being debugged must not use or reference this area. Also, the program's method of ioc allocation should be compatible with the method selected for the DEBUG run.

After the DEBUG control statement is issued, DEBUG remains in execution until an EXECUTE, STEP, or CONTINUE directive causes it to relinquish control to the user program. Control does not return to DEBUG until a user-specified breakpoint is reached during execution. When the user program terminates, DEBUG terminates also; more DEBUG directives can be processed only after another DEBUG control statement has been issued.

DEBUG CONTROL STATEMENT

The control statement that initiates execution of DEBUG is shown in figure 8-1. The parameter fname, optionally followed by fioc, must always be the first parameter; but the order of the I= and O= parameters can be reversed. If the user chooses to specify ioc numbers rather than taking advantage of automatic allocation of ioc numbers, all ioc's specified must be unique. If the defaults are used for the fioc and iioc parameters in the DEBUG control statement, fname should also use automatic ioc allocation so that there is not an overlap in ioc numbers assigned.

Sample DEBUG control statements are shown in figure 8-2.

DEBUG DIRECTIVES

The general format of each DEBUG directive is as follows. Parameters are positional and can be separated from each other and the directive name by either a blank or comma. A null parameter must be indicated by commas delimiting its position.

directive,parameter-set

DEBUG directives are listed below. Following this list, the directives are described in a logical grouping rather than in alphabetical order.

ASCII	Enter data in ASCII form
BACK	Display the data preceding the last display location
BKPT or BKPTR	Set or remove breakpoints
CONTINUE	Continue execution from the last user breakpoint
DDECIMAL	Display data in hexadecimal and decimal
DECIMAL	Enter data in decimal form
DFLOAT	Display data in hexadecimal and floating point
DISPLAY	Display data in hexadecimal and ASCII

DEBUG,fname,fioc,I=iname/iioc,O=oname/oioc/olen.

fname	Name of the existing permanent or job duration file which is the controllee file for DEBUG. Must be a virtual code file (output from the FORTRAN compiler is a virtual code file).
fioc	Optional. Input/output connector number of the controllee file, a number in the range 0 to 15 or 18 to 71. Default is an operating system assigned ioc number.
I=iname/iioc	For batch mode only, file containing the DEBUG directives:
iname	Name of an existing ASCII file of length less than or equal to 100 small pages. Default is INPUT.
iioc	Input/output connector number of iname, a number in the range 0 to 15 or 18 to 71. Default is an operating system assigned ioc number.
O=oname/oioc/olen	For batch mode, file to which all DEBUG output is written; for interactive mode, file to which data generated by the SNAP is written:
oname	Name of the file. Default is OUTPUT.
oioc	Input/output connector number of oname, a number in the range 0 to 15 or 18 to 71. Default is an operating system assigned ioc number.
olen	Length in small pages of oname, a number in the range 1 to 1000 inclusive. Default is 100.

Figure 8-1. DEBUG Control Statement Format

```
DEBUG(MYCTEE)
DEBUG(MYCTEE,10,I=MYINP,#B,O=MYOUT,12,#2C3)
DEBUG(MYCTEE,10,O=MYOUT,,#2C3,I=MYINP)
```

Figure 8-2. Sample DEBUG Control Statements

DREG	Display register contents in hexadecimal
END	Terminate execution of both DEBUG and user program
EREG	Enter hexadecimal data into a register
EXECUTE	Begin execution of user program at a specified location
FLOAT	Enter data in floating point
HEX	Enter data in hexadecimal
IDISPLAY	Display the data contained at the address found at the specified location
IDREG	Display the data found at the address specified in the given register
ROLL	Display the data following the last display location
SNAP	Dump to an output file
STAT	Provide status information such as breakpoints set, last routine referenced, last directive issued
STEP	Step through execution of user code one or more instructions at a time

Examples of DEBUG directives are shown in figure 8-3.

DI SUBR=500+4 5	If this is the first directive entered under DEBUG or if the last type referenced (if referenced at all) was S, this directive displays five words starting at location +4 words removed from location associated with statement label "500" in module SUBR. If label "500" is nonexistent in module SUBR, a message is displayed: NO SUCH LABEL OR LINE NUMBER.
HEX SUBR=500/X 10000C 880	Enter 2 half words of hexadecimal data at bit address 500 in module SUBR.
BKPT 111/L	Set a breakpoint at location that corresponds to source line number 111 in the current module. An error message is displayed if the current module is not at least 111 lines long.
DE/H 4A0/X-10	Place -10 (decimal) in the halfword at 4A0 from the beginning of the current module.
DI 0=C840/X	Display 4 words beginning at absolute virtual address C840.

Figure 8-3. Sample DEBUG Directives

Dump or Display Directives

The user can display the contents of up to 16 words of virtual memory by entering one of the following:

DISPLAY,[name=] location [/type] [±offset] [,nwords]

Displays nwords of hexadecimal and ASCII data.

DDECIMAL [/H],[name=] location [/type] [±offset] [,nwords]

Displays nwords of hexadecimal and decimal data.

DFLOAT [/H],[name=] location [/type] [±offset] [,nwords]

Displays nwords of hexadecimal and floating point data.

IDISPLAY,[name=] location [/type] [±offset] [,nwords]

Displays nwords of hexadecimal and ASCII data starting at the location indicated by the address specified by the location parameter (indirect addressing).

/H — Indicates that the data to be displayed is half-word data.

name — Name of a module, within the file, relative to which the location parameter is a reference; or 0, in which case the location is an absolute virtual address. An equals sign must immediately follow the name and precede the location, without intervening blanks, in the form name=location. Default name when DEBUG is first started is the main program (or the first module loaded, for non-FORTRAN-generated code). Otherwise, the default name is that name last referenced; in the case that the last reference was of the form 0=location, the location is assumed to be an absolute address and an associated type of S or L is disallowed.

location — A hexadecimal address, source line number, statement label, simple variable name, descriptor name, or array name, indicating location at which display is to originate or, for IDISPLAY, the location containing the address indicating the location at which display is to originate. When the offset parameter is present, the location parameter indicates a location relative to which display is to originate.

type — One of the following characters defining the type of location designated:

- S Statement label, simple variable name, descriptor name, or array name (FORTRAN programs only)
- L Source line number (FORTRAN programs only)
- X Hexadecimal bit address
- W Hexadecimal word address
- P Hexadecimal page address

Default type when DEBUG is first started is S; otherwise, the default type is that type last referenced.

offset — Hexadecimal number, indicating an upward or downward offset, in words, from the location indicated by the location parameter. A plus sign or minus sign must immediately precede the number.

nwords — Hexadecimal value designating the number of words or half-words to be displayed. Default value is #4; maximum allowed value is #10.

When the controllee file for DEBUG is a FORTRAN program that has been compiled under the S option, dynamic space fields, variables in common areas, and variables that are parameters can be displayed and altered using DEBUG directives. Variables in areas declared common can be displayed by referencing them in the module specified or last referenced. Referencing a descriptor associated with the currently allocated dynamic space fields for the breakpointed module and its higher level modules (modules that have led to the call to the breakpointed module and are linked to it through previous stack pointers) displays the contents of those fields. Variables that are parameters in the breakpointed module can be displayed by referencing them in the usual way after the prologue of the breakpointed module has been executed and the variables thereby set to their passed values; during the prologue, their values are indeterminate.

The following directives display virtual memory forward or backward from the last display location:

ROLL[,nwords] — Displays area following last location.

BACK[,nwords] — Displays area preceding last location.

nwords — Hexadecimal value designating the number of words to be displayed starting from last location displayed. Default value is the number of words specified by the previous directive; maximum allowed value is #10.

Register Directives

The user can display and alter the contents of the user program registers by issuing one of the following:

DREG,hexreg[,nregisters] — Displays the contents of a register.

EREG,hexreg,hexdata — Allows user to enter hexadecimal data into a register.

IDREG,hexreg[,nwords] Displays data found at the address that is given in specified register.

hexreg Full-word hexadecimal register number which contains data to be displayed or into which data is to be entered.

nregisters Specifies hexadecimal number of registers to be displayed, starting with hexreg. Default value is #4; maximum value allowed is #10.

hexdata Hexadecimal half-word data to be entered into n consecutive registers starting with high-order half of hexreg. Values are right-justified with zero fill.

nwords Hexadecimal value designating the number of words to be displayed. Default value is #4; maximum value allowed is #10.

Alter Memory Directives

The user can alter virtual memory by entering one of the following:

HEX,[name=] location [/type] [±offset] ,halfhex

Enters hexadecimal data.

ASCII,[name=] location [/type] [±offset] ,"ASCIIdata"

Enters an ASCII character string.

DECIMAL [/H] ,[name=] location [/type] [±offset] ,decidata

Enters decimal data.

FLOAT [/H] ,[name=] location [/type] [±offset] ,fltpt

Enters floating point data.

/H Indicates that the data to be displayed is half-word data.

name Name of a module, within the file, relative to which the location parameter is a reference; or 0, in which case the location is an absolute virtual address. An equals sign must immediately follow the name and precede the location, without intervening blanks, in the form name=location. Default name when DEBUG is first started is the main program (or the first module loaded, for non-FORTRAN-generated code). Otherwise, the default name is that name last referenced; in the case that the last reference was of the form 0=location, the location is assumed to be an absolute address and an associated type of S or L is disallowed.

location A hexadecimal address, source line number, statement label, simple variable name, descriptor name, or array name, indicating location at which data is to be entered. When the offset parameter is present, the location parameter indicates a location relative to which the data is to be entered.

type One of the following characters defining the type of location designated:

- S Statement label, simple variable name, descriptor name, or array name (FORTRAN programs only)
- L Source line number (FORTRAN programs only)
- X Hexadecimal bit address
- W Hexadecimal word address
- P Hexadecimal page address

Default type when DEBUG is first started is S; otherwise, the default type is that type last referenced.

offset Hexadecimal number, indicating an upward or downward offset, in words, from the location indicated by the location parameter. A plus sign or minus sign must immediately precede the number.

halfhex Half-word hexadecimal data values to be entered into consecutive half-word memory locations starting at location specified. Values are right-justified with zero fill.

ASCIIdata String of ASCII data to be entered into consecutive character locations starting at the position given by location parameter. The ASCII data string must be enclosed in quotation marks.

decidata Full- or half-word decimal data to be entered into consecutive full- or half-word memory locations beginning at the location specified.

fltpt Floating point data to be entered into consecutive half- or full-word memory locations, depending on data type parameters, starting at location specified. E or F format can be used.

Program Control Directives

The user can set and remove breakpoints to start and stop program execution, to dump portions of virtual memory to an output file, or to find the status of DEBUG directives issued earlier by issuing one of the following:

BKPT,[name=] location [/type] [±offset]

Defines a breakpoint. User program execution stops after the instruction located at the breakpoint address is executed.

BKPTR [,[name=] location [/type] [±offset]]

Removes a breakpoint. If no parameters are given, all breakpoints in the program are removed.

EXECUTE [,[name=] location [/type] [±offset]]

Causes DEBUG to start executing the user program at the location specified. If no parameters are given on the EXECUTE directive, DEBUG starts at the transfer address. Only one EXECUTE directive can be given per DEBUG run and it must appear before any CONTINUE directive.

CONTINUE

Causes user program execution to be continued from the last breakpoint encountered. CONTINUE can appear only after EXECUTE or another CONTINUE directive.

STEP [,ninstr]

Causes user program execution to continue for ninstr number of instructions from the last breakpoint encountered.

END

Terminates both the user program and DEBUG.

SNAP,[name=] location [/type] [±offset] [,nwords]

Dumps nwords number of words of virtual memory, starting from location, to an output file. Output data is in hexadecimal and ASCII.

SNAP,hexreg,R [,nwords]

Dumps the contents of nwords number of registers, starting with register numbered hexreg. Output is in hexadecimal and ASCII.

name — Name of a module, within the file, relative to which the location parameter is a reference; or 0, in which case the location is an absolute virtual address. An equals sign must immediately follow the name and precede the location, without intervening blanks, in the form name=location. Default name when DEBUG is first started is the main program (or the first module loaded, for non-FORTRAN-generated code). Otherwise, the default name is that name last referenced; in the case that the last reference was of the form 0=location, the location is assumed to be an absolute address and an associated type of S or L is disallowed.

location — A hexadecimal address, source line number, statement label, simple variable name, descriptor name, or array name, indicating location at which data is to be entered. When the offset parameter is present, the location parameter indicates a location relative to which the data is to be entered.

type — One of the following characters defining the type of location designated:

- S Statement label, simple variable name, descriptor name, or array name (FORTRAN programs only)
- L Source line number (FORTRAN programs only)
- X Hexadecimal bit address
- W Hexadecimal word address
- P Hexadecimal page address

Default type when DEBUG is first started is S; otherwise, the default type is that type last referenced.

offset — Hexadecimal number, indicating an upward or downward offset, in words, from the location indicated by the location parameter. A plus sign or minus sign must immediately precede the number.

ninstr — Hexadecimal number specifying a number of instructions. Default value is #1; the maximum value allowed is #10.

nwords — Hexadecimal number specifying a number of words or registers. Default value is #4.

STAT

Produces a listing of the breakpoints set, the last DEBUG and BKPT directive issued, the last routine name or common block referenced, the last type referenced, the next execution address in the user program, and displays the last module referenced.

LOOK

The LOOK utility can be used to examine statically any mass storage file to which the user has access. It cannot be used during program execution.

After LOOK execution has been initiated with the LOOK control statement, the utility responds to directions received from directives. The modifications made through LOOK directives persist for the life of the file modified. LOOK remains in execution until an END is received.

LOOK CONTROL STATEMENT

The control statement that initiates execution of LOOK is shown in figure 8-4. The parameter fname must always be the first parameter, but the order of the I= and L= parameters can be reversed.

LOOK,fname,I=iname,L=oname/olen/disp.

fname	Name of the existing mass storage file being modified or examined by LOOK.
I=iname	Optional. Name of an existing ASCII file containg directives for LOOK. Default is I=INPUT.
L=oname/olen/disp	Optional. For batch mode, describes the file to which all LOOK output is written. For interactive mode, describes the file to which output is written when the PRINT directive is issued:
	oname — Name of the file. Default is OUTPUT.
	olen — Integral length of oname in small pages, in decimal; must be greater than 0 and less than 1001. Default is 128 small pages.
	disp — Disposition of oname, currently PR only. Default is PR.
	If oname exists already, it is destroyed before being created.

Figure 8-4. LOOK Control Statement Format

PRINT/PAGE/HEX,0,8

PRINT indicates that all successive output is to be written to the file OUTPUT. PAGE indicates that all addresses entered subsequently are page addresses. HEX,0,8 causes a dump of 8 full words from the beginning of the first page in the file.

HEX,,8 — Display 8 words at the beginning of the first page in the file, assuming the previous PAGE directive. The three columns displayed indicate the word address, hexadecimal contents, and contents in ASCII.

SEARCH,'/',,3 — Search the file for the third appearance of the character /, beginning at the start of the file. The address at which the specified occurrence is found is reported on the output file.

BIT/EASCII,C0,'1111'/HEX,C0,1

BIT indicates that all addresses entered subsequently are bit addresses. EASCII places the ASCII character string 1111 at the halfword address C0. The next directive displays the word just entered in hexadecimal and ASCII.

Figure 8-5. Sample LOOK Directives

LOOK DIRECTIVES

All numbers in all LOOK directives, unless otherwise specified, must be hexadecimal. Directives can be concatenated by slashes. For example, V/W/HEX,1000,10 is valid, and it indicates three directives that LOOK would process consecutively, just as if they had been issued separately in the same order.

When the parameters for a given directive are meaningless, missing, or illegal, the directive is ignored and an error message is issued.

The response format for the directive is:

COMMAND=command keyword
output

Examples of LOOK directives are shown in figure 8-5.

In the individual descriptions given below, optional parameters can be omitted and a default value will be assigned. If an embedded parameter is to be omitted, commas must be used to maintain positional integrity; for example, HEX,,2 defaults the beginning address to 0. If the last one or more parameters are to be omitted, the command must end with the last specified parameter. For example, HEX defaults address to 0 and length to 1; HEX,100 defaults length to 1.

LOOK directives are listed below. Following this list, the directives are described in a logical grouping rather than in alphabetical order.

BACK	Display file portion that immediately precedes portion last displayed.
BASE	All addresses in other LOOK directives are to be offset by a specified amount.
BIT	All addresses in other LOOK directives are to be interpreted as bit addresses.
DEC	Dump or display file in full-word decimal format.
DISPLAY	All output is to go to the terminal.
EASCII	Enter ASCII character string in file at specified location.
EDEC	Enter full-word decimal data items into file at specified location.
EFLOAT	Enter full-word floating point data items into file at specified location.
EHDEC	Enter half-word decimal data items into file at specified location.
EHEX	Enter half-word hexadecimal data items into file at specified location.

EHFLOAT Enter half-word floating point data items into file at specified location.

END End LOOK.

FLOAT Dump or display file in full-word floating point format.

HDEC Dump or display file in hexadecimal and half-word decimal format.

HEX Dump or display file in hexadecimal and ASCII format.

HFLOAT Dump or display file in hexadecimal and half-word floating point format.

IDEC Dump or display file in full-word decimal format (indirect addressing of file).

IFLOAT Dump or display file in hexadecimal and half-word floating point format (indirect addressing of file).

IHDEC Dump or display file in hexadecimal and half-word decimal format (indirect addressing of file).

IHEX Dump or display file in hexadecimal and ASCII format (indirect addressing of file).

IHFLOAT Dump or display file in hexadecimal and half-word floating point format (indirect addressing of file).

PAGE All addresses in other LOOK directives are to be interpreted as page addresses.

PATTERN Enter pattern into portion of file.

PRINT All output from directives is to go to the output file.

ROLL Display file portion that immediately follows portion last displayed.

SEARCH Search file for appearance n of a specified character string.

SEQ All addresses in other LOOK directives are to be interpreted as sequential from this point on.

VIRTUAL All addresses in other LOOK directives are to be interpreted as being virtual.

WORD All addresses in other LOOK directives are to be interpreted as word addresses.

SEARCH Directive

LOOK searches a file for an occurrence of a specified string of characters when the SEARCH directive is issued:

SEARCH,'string', [addr],[n]

string A string of ASCII characters. The string must be enclosed in single quotes.

addr Position in file where search for the character string is to begin, designated by a byte address. If addr does not lie on a byte boundary, it is truncated to the nearest byte boundary. Default is 0.

n Integer constant greater than zero, indicating the occurrence of the character string that is to be selected. Default is 1.

Beginning at addr, the file is searched for n occurrences of string, and the address of the last is returned. If LOOK is in virtual mode, the search is through the file virtually. If LOOK is in sequential mode, the search is through the file sequentially.

If no occurrences of the character string are found, the message

CHARACTER STRING NOT FOUND

is issued. If m, but fewer than n, occurrences are found when the end of the file is reached, the message

ONLY m OCCURRENCES WERE FOUND

is issued. If the search is successful, a message

CHARACTER STRING FOUND AT address

is issued, where address is the bit address of the first character of the selected string.

Disposition of Directive Output

The interactive user can select whether output is to be displayed at the terminal or dumped onto the output file that was specified in the LOOK control statement. Initial mode is DISPLAY. The directives that control the disposition of output are:

PRINT This indicates that from the time of this directive, or until a DISPLAY or END directive is entered, all output is to be written to the output file.

DISPLAY This indicates that from the time of this directive, or until a PRINT or END directive is entered, all output is to be sent to the terminal.

END End LOOK.

Display and Dump Directives

The following directives cause dump or display of a specified number of words of the file starting at a specified location in the file. For interactive users in display mode, the display is one word or half-word per line. On the output file, four words are placed in one line, with duplicate lines being suppressed. The directives are:

DEC,[addr],[len] Displays/dumps file portion as hexadecimal and full-word decimal data.

FLOAT,[addr],[len] Displays/dumps file portion as hexadecimal and full-word floating point data.

HDEC,[haddr],[len] Displays/dumps file portion as hexadecimal and half-word decimal data.

HEX,[haddr],[len] Displays/dumps file portion as hexadecimal and ASCII data.

HFLOAT,[haddr],[len] Displays/dumps file portion as hexadecimal and half-word floating point data.

addr Position in file where display/dump is to begin, designated by a word address. If addr is not on a word boundary, it is truncated to the nearest word boundary. Default is 0.

haddr Position in file where display/dump is to begin, designated by a half-word address. If haddr is not on a half-word boundary, it is truncated to the nearest half-word boundary. Default is 0.

len The number of words displayed/dumped. If len is not specified, it is taken to be 1.

Corresponding to each of the above directives is another LOOK directive that performs the identical operation, except that the address parameter specified must be the indirect, rather than the direct, address of the position where display or dump is to begin. The directives are:

IDEC,[addr],[len] Displays/dumps file portion as hexadecimal and full-word decimal data.

IFLOAT,[addr],[len] Displays/dumps file portion as hexadecimal and full-word floating point data.

IHDEC,[addrh],[len] Displays/dumps file portion as hexadecimal and half-word decimal data.

IHEX,[addrh],[len] Displays/dumps file portion as hexadecimal and ASCII data.

IHFLOAT,[addrh],[len] Displays/dumps file portion as hexadecimal and half-word floating point data.

addr Address of word containing position in file where display/dump is to begin; if addr is not a full-word address, the value of addr is interpreted to be the first word boundary preceding addr. The low 48 bits of the word at addr is the file position where the display or dump begins; if the 48 bits do not constitute a word address, the display/dump begins on the first word boundary preceding the address. Default is 0.

haddr Address of word containing position in file where display/dump is to begin; if haddr is not a full-word address, the value of haddr is interpreted to be the first word boundary preceding haddr. The low 48 bits of the word at haddr is the file position where the display or dump begins; if the 48 bits do not constitute a half-word address, the display/dump begins on the first halfword boundary preceding the address. Default is 0.

len The number of words displayed/dumped. If len is not specified, it is taken from the top 16 bits of the word at addr or haddr. Default is 1.

Additional data can be displayed or dumped with either of the following:

ROLL Displays or dumps the file portion that immediately follows the portion last displayed or dumped. The number of words and format is the same as that of the previous display/dump.

BACK Displays or dumps the file portion that immediately precedes the portion last displayed or dumped. The number of words and format is the same as that of the previous display/dump.

Directives for Entering Values

LOOK offers seven directives for entering values into the file. The directives cause values to be placed in the file beginning at a particular location. The operation performed is not an insertion; that is, does not cause expansion of the size of the file, but is, instead, a replacement of the current value with another.

Any file on which these operations are performed must have write access. The directives are:

PATTERN,haddr,laddr,$data_0$[,$data_1$. . .,$data_n$]

Enters half-word data pattern into the file between haddr and laddr inclusive.

EASCII,haddr,'string'

Enters character string starting at haddr.

EDEC,haddr,$data_0$[,$data_1$. . .,$data_n$]

Enters full-word decimal data items starting at haddr.

EFLOAT,addr,$data_0$[,$data_1$. . .,$data_n$]

Enters full-word floating point data items starting at addr.

EHDEC,haddr,$data_0$[,$data_1$. . .,$data_n$]

Enters half-word decimal data items starting at haddr.

EHEX,haddr,$data_0$[,$data_1$. . .,$data_n$]

Enters half-word hexadecimal data starting at haddr.

EHFLOAT,haddr,$data_0$[,$data_1$. . .,$data_n$]

Enters half-word floating point data items starting at haddr.

haddr Position in file where entry is to begin, designated by a hexadecimal half-word address; haddr must be on a half-word boundary.

laddr Position in file where the last half-word entered is to be placed, designated by a hexadecimal half-word address; laddr must be on a half-word boundary.

$data_i$ The data to be entered; either a half-word or a full-word of data, depending on the directive. Depending on the directive, $data_i$ is a hexadecimal, decimal, or floating point number that can be represented in a half-word or full-word.

addr Position in file where entry is to begin, designated by a hexadecimal full-word address; addr must be on a full-word boundary.

string A string of ASCII characters. The string must be enclosed in single quotes. If the number of characters in the string is not a multiple of 4, the last half-word is blank filled on the right.

Declaration of Directive Address Type

The user can indicate whether addresses in LOOK directives specify a quantity of bits, words, or pages by entering one of the following:

WORD Specifies that all addresses in LOOK directives are to be interpreted as word addresses.

PAGE Specifies that all addresses in subsequent LOOK directives are to be interpreted as small page addresses.

BIT Specifies that all addresses in subsequent LOOK directives are to be interpreted as bit addresses.

The user can indicate that all addresses in LOOK directives are to be offset by entering:

BASE,offset

offset Offset to be added to all addresses. The offset is in bits, words, or page, depending on the address mode established by BIT, WORD, or PAGE directives.

Initially, BASE is 0.

The user can indicate whether the virtual file being manipulated is to be accessed as a virtual or physical file by entering one of the directives:

SEQ Declares that from this point on, or until VIRTUAL or END is entered, all addresses referred to in other directives are sequential addresses. This allows access to the one or two minus pages of a virtual file, which are not virtually addressable.

VIRTUAL Declares that from this point on, or until SEQ or END is entered, all addresses referred to in other LOOK directives are virtual addresses.

For physical files, these directives are meaningless and if entered are ignored; physical files are always in SEQ mode. At the beginning of a LOOK run, mode is VIRTUAL (unless the file is physical).

These specifications hold until another of these directives is entered or LOOK ends. Initial address mode is BIT.

DUMP

When a fatal error occurs during execution of a program, a dump of information extracted mostly from the drop file for the executing program can be placed in the output file for the program. This dump is performed automatically by the batch processor. The standard items in the dump are the following, in the indicated order:

Program address.

Last executed instruction in hexadecimal.

System error code of the fatal error condition and the address indicating where the error occurred (also returned in word 139 of the first minus page). Codes are defined in volume 2.

Subroutine traceback. If the error occurred in a subroutine, the address of each CALL statement that led to that subroutine is listed.

List of the open (active) files at the time the error occurred along with each file's virtual page address and its length in pages.

Alpha and Beta words if the word preceding the program address contains an exit force instruction.

Contents of the first minus page in hexadecimal and ASCII. Duplicate lines are suppressed.

Contents of the second minus page in hexadecimal and ASCII.

Contents of memory in hexadecimal and ASCII, from 50 words preceding the program address to 50 words following the address.

If the error occurred in a subroutine, the register save area for each caller is dumped, followed by the contents of memory from 50 words preceding to 50 words following the caller's address in the subroutine traceback list. Dumps are output in reverse sequence of the CALL occurrences that led to the failing subroutine. Subroutines are always dumped, regardless of whether they are system-supplied or user-supplied.

On some fatal error conditions, no dump is made automatically, in which case the user has the option of requesting the dump if the drop file for the program has persisted. The FILES command can be issued to determine if the drop file has been retained automatically as one of the user's private files and, if the file has been retained, to determine the drop file name for use in the DUMP control statement.

The format of the DUMP control statement is given in figure 8-6. It can be issued either interactively or from within a batch control sequence.

```
DUMP, dropfile.

dropfile    Name of program's drop file.
```

Figure 8-6. DUMP Control Statement Format

CHARACTER SET

The ASCII character set is shown in table A-1. Aids for hexadecimal-to-octal and hexadecimal-to-decimal conversion are given in tables A-2 and A-3.

TABLE A-1. AMERICAN NATIONAL STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII) WITH PUNCHED CARD CODES AND EBCDIC TRANSLATION

	b_8 b_7 b_6 b_5	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
b_4 b_3 b_2 b_1	COL / ROW	0	1	2	3	4	5	6	7	8	9	10 (A)	11 (B)	12 (C)	13 (D)	14 (E)	15 (F)
0 0 0 0	0	NUL 12-0-9-8-1 NUL 00	DLE 12-11-9-8-1 DLE 10	SP no-punch SP 40	0 0 0 F0	@ 8-4 @ 7C	P 11-7 P D7	` 8-1 ` 79	p 12-11-7 p 97	12-11-0-9-8-1 DS 20	12-11-0-9-8-1 30	12-0-9-1 41	12-11-9-8 58	12-11-0-9-6 76	12-11-8-7 9F	12-11-0-8 B8	12-11-9-8-4 DC
0 0 0 1	1	SOH 12-9-1 SOH 01	DC1 11-9-1 DC1 11	! 12-8-7 4F	1 1 1 F1	A 12-1 A C1	Q 11-8 Q D8	a 12-0-1 a 81	q 12-11-8 q 98	0-9-1 SOS 21	9-1 31	12-0-9-2 42	11-8-1 59	12-11-0-9-7 77	11-0-8-1 A0	12-11-0-9 B9	12-11-9-8-5 DD
0 0 1 0	2	STX 12-9-2 STX 02	DC2 11-9-2 DC2 12	" 8-7 " 7F	2 2 2 F2	B 12-2 B C2	R 11-9 R D9	b 12-0-2 b 82	r 12-11-9 r 99	0-9-2 FS 22	11-9-8-2 CC 1A	12-0-9-3 43	11-0-9-2 62	12-11-0-9-8 78	11-0-8-2 AA	12-11-0-8-2 BA	12-11-9-8-6 DE
0 0 1 1	3	ETX 12-9-3 ETX 03	DC3 11-9-3 TM 13	# 8-3 # 7B	3 3 3 F3	C 12-3 C C3	S 0-2 S E2	c 12-0-3 c 83	s 11-0-2 s A2	0-9-3 23	9-3 33	12-0-9-4 44	11-0-9-3 63	12-0-8-1 80	11-0-8-3 AB	12-11-0-8-3 BB	12-11-9-8-7 DF
0 1 0 0	4	EOT 9-7 EOT 37	DC4 9-8-4 DC4 3C	$ 11-8-3 $ 5B	4 4 4 F4	D 12-4 D C4	T 0-3 T E3	d 12-0-4 d 84	t 11-0-3 t A3	0-9-4 BYP 24	9-4 PN 34	12-0-9-5 45	11-0-9-4 64	12-0-8-2 8A	11-0-8-4 AC	12-11-0-8-4 BC	11-0-9-8-2 EA
0 1 0 1	5	ENQ 0-9-8-5 ENQ 2D	NAK 9-8-5 NAK 3D	% 0-8-4 % 6C	5 5 5 F5	E 12-5 E C5	U 0-4 U E4	e 12-0-5 e 85	u 11-0-4 u A4	11-9-5 NL 15	9-5 RS 35	12-0-9-6 46	11-0-9-5 65	12-0-8-3 8B	11-0-8-5 AD	12-11-0-8-5 BD	11-0-9-8-3 EB
0 1 1 0	6	ACK 0-9-8-6 ACK 2E	SYN 9-2 SYN 32	& 12 & 50	6 6 6 F6	F 12-6 F C6	V 0-5 V E5	f 12-0-6 f 86	v 11-0-5 v A5	12-9-6 LC 06	9-6 UC 36	12-0-9-7 47	11-0-9-6 66	12-0-8-4 8C	11-0-8-6 AE	12-11-0-8-6 BE	11-0-9-8-4 ⌐ EC
0 1 1 1	7	BEL 0-9-8-7 BEL 2F	ETB 0-9-6 ETB 26	' 8-5 ' 7D	7 7 7 F7	G 12-7 G C7	W 0-6 W E6	g 12-0-7 g 87	w 11-0-6 w A6	11-9-7 IL 17	12-9-8 GE 08	12-0-9-8 48	11-0-9-7 67	12-0-8-5 8D	11-0-8-7 AF	12-11-0-8-7 BF	11-0-9-8-5 ED
1 0 0 0	8	BS 11-9-6 BS 16	CAN 11-9-8 CAN 18	(12-8-5 (4D	8 8 8 F8	H 12-8 H C8	X 0-7 X E7	h 12-0-8 h 88	x 11-0-7 x A7	0-9-8 28	9-8 38	12-8-1 49	11-0-9-8 68	12-0-8-6 8E	12-11-0-8-1 B0	12-0-9-8-2 CA	11-0-9-8-6 EE
1 0 0 1	9	HT 12-9-5 HT 05	EM 11-9-8-1 EM 19	) 11-8-5) 5D	9 9 9 F9	I 12-9 I C9	Y 0-8 Y E8	i 12-0-9 i 89	y 11-0-8 y A8	0-9-8-1 29	9-8-1 39	12-11-9-1 51	0-8-1 69	12-0-8-7 8F	12-11-0-1 B1	12-0-9-8-3 CB	11-0-9-8-7 EF
1 0 1 0	10 (A)	LF 0-9-5 LF 25	SUB 9-8-7 SUB 3F	* 11-8-4 * 5C	: 8-2 : 7A	J 11-1 J D1	Z 0-9 Z E9	j 12-11-1 j 91	z 11-0-9 z A9	0-9-8-2 SM 2A	9-8-2 3A	12-11-9-2 52	12-11-0 70	12-11-8-1 90	12-11-0-2 B2	12-0-9-8-4 ♪ CC	12-11-0-9-8-2 ¦ (LVM) FA
1 0 1 1	11 (B)	VT 12-9-8-3 VT 0B	ESC 0-9-7 ESC 27	+ 12-8-6 + 4E	; 11-8-6 ; 5E	K 11-2 K D2	[12-8-2 ¢ 4A	k 12-11-2 k 92	{ 12-0 { C0	0-9-8-3 CU2 2B	9-8-3 CU3 3B	12-11-9-3 53	12-11-0-9-1 71	12-11-8-2 9A	12-11-0-3 B3	12-0-9-8-5 CD	12-11-0-9-8-3 FB
1 1 0 0	12 (C)	FF 12-9-8-4 FF 0C	FS 11-9-8-4 IFS 1C	, 0-8-3 , 6B	< 12-8-4 < 4C	L 11-3 L D3	\ 0-8-2 \ E0	l 12-11-3 l 93	¦ 12-11 ¦ 6A	0-9-8-4 2C	12-9-4 PF 04	12-11-9-4 54	12-11-0-9-2 72	12-11-8-3 9B	12-11-0-4 B4	12-0-9-8-6 Ψ CE	12-11-0-9-8-4 FC
1 1 0 1	13 (D)	CR 12-9-8-5 CR 0D	GS 11-9-8-5 IGS 1D	– 11 – 60	= 8-6 = 7E	M 11-4 M D4	] 11-8-2 ! 5A	m 12-11-4 m 94	} 11-0 } D0	12-9-8-1 RLF 09	11-9-4 RES 14	12-11-9-5 55	12-11-0-9-3 73	12-11-8-4 9C	12-11-0-5 B5	12-0-9-8-7 CF	12-11-0-9-8-5 FD
1 1 1 0	14 (E)	SO 12-9-8-6 SO 0E	RS 11-9-8-6 IRS 1E	. 12-8-3 . 4B	> 0-8-6 > 6E	N 11-5 N D5	^ 11-8-7 ¬ 5F	n 12-11-5 n 95	~ 11-0-1 ~ A1	12-9-8-2 SMM 0A	9-8-6 3E	12-11-9-6 56	12-11-0-9-4 74	12-11-8-5 9D	12-11-0-6 B6	12-11-9-8-2 DA	12-11-0-9-8-6 FE
1 1 1 1	15 (F)	SI 12-9-8-7 SI 0F	US 11-9-8-7 IUS 1F	/ 0-1 / 61	? 0-8-7 ? 6F	O 11-6 O D6	_ 0-8-5 _ 6D	o 12-11-6 o 96	DEL 12-9-7 DEL 07	11-9-8-3 CU1 1B	11-0-9-1 E1	12-11-9-7 57	12-11-0-9-5 75	12-11-8-6 9E	12-11-0-7 B7	12-11-9-8-3 DB	EO 12-11-0-9-8-7 FF

64-Character ASCII Subset

96-Character ASCII Subset

LEGEND

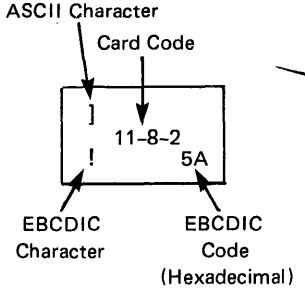
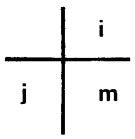


TABLE A-2. HEXADECIMAL-OCTAL CONVERSION

Second Hexadecimal Digit (Rightmost of a 2-digit number)	First Hexadecimal Digit (Leftmost of a 2-digit number) 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	000	020	040	060	100	120	140	160	200	220	240	260	300	320	340	360
1	001	021	041	061	101	121	141	161	201	221	241	261	301	321	341	361
2	002	022	042	062	102	122	142	162	202	222	242	262	302	322	342	362
3	003	023	043	063	103	123	143	163	203	223	243	263	303	323	343	363
4	004	024	044	064	104	124	144	164	204	224	244	264	304	324	344	364
5	005	025	045	065	105	125	145	165	205	225	245	265	305	325	345	365
6	006	026	046	066	106	126	146	166	206	226	246	266	306	326	346	366
7	007	027	047	067	107	127	147	167	207	227	247	267	307	327	347	367
8	010	030	050	070	110	130	150	170	210	230	250	270	310	330	350	370
9	011	031	051	071	111	131	151	171	211	231	251	271	311	331	351	371
A	012	032	052	072	112	132	152	172	212	232	252	272	312	332	352	372
B	013	033	053	073	113	133	153	173	213	233	253	273	313	333	353	373
C	014	034	054	074	114	134	154	174	214	234	254	274	314	334	354	374
D	015	035	055	075	115	135	155	175	215	235	255	275	315	335	355	375
E	016	036	056	076	116	136	156	176	216	236	256	276	316	336	356	376
F	017	037	057	077	117	137	157	177	217	237	257	277	317	337	357	377
Octal	000 – 037		040 – 077		100 – 137		140 – 177		200 – 237		240 – 277		300 – 337		340 – 377	

TABLE A-3. HEXADECIMAL-DECIMAL CONVERSION

		Exponent for Base 16					
		5	4	3	2	1	0
Hexadecimal Number	0	0	0	0	0	0	0
	1	1048576	65536	4096	256	16	1
	2	2097152	131072	8192	512	32	2
	3	3145728	196608	12288	768	48	3
	4	4194304	262144	16384	1024	64	4
	5	5242880	327680	20480	1280	80	5
	6	6291456	393216	24576	1536	96	6
	7	7340032	458752	28672	1792	112	7
	8	8388608	524288	32768	2048	128	8
	9	9437184	589824	36864	2304	144	9
	A	10485760	655360	40960	2560	160	10
	B	11534336	720896	45056	2816	176	11
	C	12582912	786432	49152	3072	192	12
	D	13631488	851968	53248	3328	208	13
	E	14680064	917504	57344	3584	224	14
	F	15728640	983040	61440	3840	240	15



$j_{16} \times 16^{i} = m_{10}$

To find $E_{16} \times 16^{3}$; look at row E, column 3 and find 57344

Table B-1 describes the error codes produced by the tasks that are collectively known as USER1. The codes are returned to any terminal logged in with the operator's user number. System messages are described in volume 2; see the error response field (ss, cerr, or serr field) of any message specified in the Significance column. Resident system function codes are described in internal documentation. Error 21 is a warning that requires no user action. For errors 10, 17, 20, and 23, possibly the user can take corrective measures; for other errors, see a system analyst.

Table B-2 describes System Record Manager error codes.

Table B-3 lists the messages produced by utilities or other system features described in this manual. USER1 errors detected by CARDREDR (also defined in table B-3) are printed in a user dayfile and are not returned to the operator's terminal. The return codes are those that can be tested by a TV control statement in a batch job. Code interpretation is:

0 No error

4 Warning (nonfatal) errors

8 Fatal error and task abort

If the issuing task name is Q7KEYWRD or Q7GETFIL, the message was produced by a task the utilities use for checking syntax or accessing files. PF indicates the message was issued by the permanent file utilities. Messages are displayed at the terminal of an interactive user or on the dayfile of a batch job.

Table B-4 lists System Request Language error messages.

TABLE B-1. USER1 ERROR CODES

Error Code	Significance	Resident System Function
1-00	Unable to lock down memory to allow I/O file transfer.	F00D, F004
2-00	Unable to unlock memory previously locked down for I/O file transfer.	F003
3-00	Unable to read print or punch file from mass storage.	C500
4-00	Unable to create print or punch file on Service Station device.	C312
5-00	Unable to open file located on Service Station device.	C312
6-00	Error during writing of output file to Service Station.	C301
6-50	Error during reading of an output file that was just written to the Service Station.	C300
7-xx	Error during closing of card input file. See system message CLOSE FILE for explanation of xx.	
8-00	Unable to close Service Station input or output file.	C312
10-xx	Error during creation of card input file. See system message CREATE FILE for explanation of xx.	
11-00	Error during reading of input file from Service Station device.	C300
12-00	Error during removing of input file from Service Station queue.	C311
13-00	Error during destroying of input file located on Service Station.	C312
14-xx	Error during destroying of a file that could not be given to a user. See system message DESTROY FILE for explanation of xx.	
15-00	Error returned on a LIST FILE INDEX OR SYSTEM TABLE message during execution of the PRINTOUT or PUNCHOUT routine.	
16-xx	Error during destroying of an output file after it was processed. See system message DESTROY FILE for explanation of xx.	

TABLE B-1. USER1 ERROR CODES (Contd)

Error Code	Significance	Resident System Function
17-xx	Batch processor was not started. See system message EXECUTE PROGRAM FOR USER NUMBER for explanation of xx.	
18-00	Unable to queue output file on Service Station.	C311
19-00	Error during transferral of card input file to mass storage.	C501
20-xx	Error during the giving of an input file to its owner. See system message GIVE FILE for explanation of xx.	
21-02	Warning. Reduce not possible for Service Station file; the full length of the file is used.	C312
21-04	Warning. PRINTOUT has been given a family of files with too many members; the family is printed in parts.	
21-05	Warning. PRINTOUT has been given a file, or a family of files, that has a length greater than the maximum allowed; this file or family of files is printed in parts.	
21-06	Warning. CLOSE FILE error in PUNCHOUT routine.	
22-00	Zip codes could not be found during output processing of a LIST FILE INDEX OR SYSTEM TABLE message.	
23-00	Invalid internal characteristic field for LIST FILE INDEX OR SYSTEM TABLE message (PRINTOUT only).	
24-xx	Requeuing is not possible for an output file. See system message GIVE FILE for explanation of xx.	
25-00	Error during OPEN of a file during execution of the POSTOUT or PRINTOUT routine.	
26-xx	Error during creation of a file for a card input file and during destroying of the Service Station copy of the file. See system message CREATE FILE for explanation of xx.	C312
27-00	Error during closing of an output file.	
28-xx	Error during changing of the account number of the input file. See system message CHANGE FILE NAME OR ACCOUNT for explanation of xx.	
29-00	Reserved for CYBERIN. See CYBER 200 Link Operator's Guide.	
29-xx	Error during destroying of an existing file having the same name as the input file. See system message DESTROY FILE for explanation of xx.	
3x-xx	Reserved for CYBERIN. See CYBER 200 Link Operator's Guide.	
4x-xx	Reserved for CYBEROUT. See CYBER 200 Link Operator's Guide.	
5x-xx	Reserved for CYBERIN. See CYBER 200 Link Operator's Guide.	
60-xx	Error during creation of an error dayfile (as created by CARDREDR). See system message CREATE FILE for explanation of xx.	
61-xx	Error during closing of an error dayfile (as created by CARDREDR). See system message CLOSE FILE for explanation of xx.	
62-xx	Error during routing of the error dayfile (as created by CARDREDR). See system message ROUTE AND FILE DISPOSITION for explanation of xx.	
63-xx	Error during opening of a punch file. See system message OPEN FILE for explanation of xx.	

TABLE B-1. USER1 ERROR CODES (Contd)

Error Code	Significance	Resident System Function
68-00	Error in trying to read an output file that was just written to the Service Station.	C300
69-xx	Unable to destroy a Service Station file after USER1 error xx.	C312
92-xx	Error during a ROUTE call in the CARDREDR routine. See system message ROUTE AND FILE DISPOSITION for explanation of xx.	
93-01	Error during the destroying of a Service Station file during execution of the POSTOUT routine.	C312
94-01	Error returned on a LIST FILE INDEX OR SYSTEM TABLE message during execution of the POSTOUT routine.	
95-01	Service Station create file error during execution of the POSTOUT routine.	C312
96-01	Mass storage read file error during execution of the POSTOUT routine.	C500
97-01	Service Station write file error during execution of the POSTOUT routine.	C301
98-01	Service Station close file error during execution of the POSTOUT routine.	C312
99-01	Service Station queueing error during execution of the POSTOUT routine.	C312

TABLE B-2. SYSTEM RECORD MANAGER ERROR CODES

Error Code	Significance	Error Code	Significance
0000	Normal completion of last SRM request.	0015	Pages have already been written to the existing drop file; no new file was created for user-generated drop file.
0001	Invalid SRM call or calling parameter.		
0002	Beta buffer length error or too many Beta requests.	0016	Drop file length for user-generated drop file is not adequate to hold space already in drop file map for DEFINE.
0003	File already exists.		
0004	No available mass storage space for this file.	0017	Unable to find requested packid.
0005	MCAT is illegal.	0018	Cannot send tape request message to operator.
0006	Parameter of format error occurred.	0019	Bound implicit map area in minus page is full.
0007	Operator-initiated tape error occurred.	0020	Virtual address overlap.
0008	Input/output connector is already in use or not available or not valid.	0021	Error from file; could not find a slot in PFI or FILEI.
0009	File index is full.	0022	Could not make a PFI entry.
0010	Standby job may not create a tape.	0023	Tried to create a virtual file but file is less than two pages long.
0011	Invalid file name.	0024	Disk is logically turned off.
0012	SS was preset.	0025	Cannot locate user or pool for privileged DEFINE.
0013	Existing drop file cannot be destroyed; DMAP full or file open to another task.	0026	No room for file entries for privileged DEFINE.
0014	Access code or file type is illegal.		

TABLE B-2. SYSTEM RECORD MANAGER ERROR CODES (Contd)

Error Code	Significance
0027	Either no name was given, or the name was not in the file index.
0028	Mass storage error occurred while reading a virtual file minus page.
0029	File requested has a security classification higher than that of the caller.
0030	BVA + BLENGTH is greater than maximum user virtual address for OPEN FILE.
0031	User directory not found or pool not found for privileged OPEN.
0032	File has been privileged-opened by another caller; access not allowed.
0033	Nonprivileged user.
0034	Format error in privileged OPEN.
0035	No more write opens permitted.
0036	PTR points to a file index copy that is out of bounds for privileged OPEN.
0037	No more room for user table for privileged OPEN.
0038	More than one virtual address overlap; usually indicates an uninitialized minus page.
0039	Input/output connector was not for a mass storage file or not open for explicit I/O.
0040	Attempt to alter a public file index entry; input/output connector is cleared, but no information is altered in the file index (file closed).
0041	File whose CLOSE was requested, but a file page is still locked down.
0042	A scratch or output file is open to another program of this user (file closed to this task but not destroyed or given).
0043	Invalid name for a file with a management category of output (file closed).
0044	Specified ioc was not open for CLOSE.
0045	Drop file map full.
0046	Format error in privileged CLOSE.
0047	Illegal interrupt address.
0048	Buffer size is greater than 24 small pages or is zero pages for a small buffer definition request.
0049	Illegal tape density requested.
0050	Illegal operation or sub-operation for explicit I/O.
0051	Illegal TPM or mode field for explicit I/O.
0052	No buffer assigned for explicit I/O.
0053	Page address specified for explicit I/O was out-of-bounds.
0054	Illegal attempt to access a read only or write only file.
0055	Interrupt requested, but no interrupt address specified for explicit I/O.
0056	Request for greater than 128 small pages for a large page buffer definition request.
0057	Buffer crosses a large page boundary for a large buffer.
0058	Attempt was made to open a buffer that was already open.
0059	Buffer already in use. Previous input/output which uses the same buffer is not complete.
0060	Attempt to reuse Alpha before the previous call, which uses the same Alpha address, is complete.
0061	Device not ready.
0062	Tape, SBU, or transmission parity error.
0063	Data exceeds user buffer.
0064	End of tape.
0065	End of file.
0066	Attempt to write on file-protected tape.
0067	Channel failure (disk only).
0068	Lost data on tape record.
0069	Attempted backspace at load point.
0070	Error in positioning mass storage device.
0071	Explicit I/O request aborted by station operator.
0072	Multiple station errors occurred during last explicit I/O request. A call to the STATUS routine is required to determine the error bits set in the serr field.

TABLE B-2. SYSTEM RECORD MANAGER ERROR CODES (Contd)

Error Code	Significance	Error Code	Significance
0073	File is still open to an active program.	0082	No such user number exists.
0074	Tried to reduce a virtual file to less than two small-page blocks.	0083	User number specified is that of the public list for GIVE.
0075	File name does not exist.	0084	File to be given is a source or drop file.
0076	File name given is in conflict with that in the input/output connector.	0085	Recipient of a file to be given has a security classification less than that of the file.
0077	Format error in DESTROY request.		
0078	Nonprivileged task tried to destroy a public file.	0086	Pool name specified does not exist, or giver is not a member of this pool.
0079	User other than pool boss has tried to destroy a pool file.	0087	File to be changed is still active.
0080	Error trying to remove PFI entry during DESTROY request.	0088	New account number is not valid for CHANGE.
0081	File has the same name as a public file.	0089	New length specified for the file is greater than the existing length for REDUCE.

TABLE B-3. DIAGNOSTIC MESSAGES

Message	Significance/Action	Return Code	Issued By
ACCESS ATTEMPTED IS NOT ALLOWED	File access field does not allow this access. Correct TCOPY directive.	8	TCOPY
ADDR=REGnn	Address indicated by the symbol specified corresponds to register nn.	4	DEBUG
ADR IN REG FILE	A location, or the computation of name= location, was less than #4000.	4	DEBUG
ADR NOT COD ADR	Address specified in EXECUTE, BKPT, or BKPTR command is not that of executable code. No action will be taken.	4	DEBUG
ADR BEYOND MOD	Address specified does not fall within the current module.	4	DEBUG
ALPHA OUT OF BOUNDS	Address of Alpha message is out-of-bounds.	-	Op System
ALREADY ATTACHED TO 4 POOLS	User can be attached to only four pools simultaneously.	8	PATTACH
ATTEMPT TO EXTEND PAST MAX ON FILE	Original length of file too small. Rerun with a larger file.	-	Op System
ATTEMPT TO READ PAST EOF ON FILE	Do not attempt to read past end of file.	-	Op System
ATTEMPTED TO CALL NON-COMDECK	Correct CALL statement	4	UPDATE

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
ATTEMPTED TO READ FROM UNKNOWN FILE	Correct READ statement.	8	UPDATE
BACKSPACE AT LOAD POINT	Correct TCOPY directive.	8	TCOPY
BACKSPACE BEYOND BEGINNING OF FILE	Informative message.	0	TCOPY
BAD ACCOUNT	Invalid account on a LOGON statement.	-	Op System
BAD BINARY FILE lfn THE WORD 'MODULE' NOT FOUND	Self-explanatory.	8	LOAD
BAD CARD ENCOUNTERED	Correct input file card.	4	UPDATE
BAD CLASS	Class specified was not I, B, P or S.	-	Op System
BAD COMMAND	Correct DEBUG control statement.	4	DEBUG
BAD DFM IN CTEE	The controllee to be debugged had been loaded with only a portion of the Data Flag Manager. When the controllee contains DFM, DEBUG links to the controllee's copy of DFM. When the controllee does not contain DFM, DEBUG uses its own copy. However, when the controllee contains only part of the DFM, DEBUG assumes something is wrong and aborts. The controllee should contain all or none of the following module/entry point: FT _ SYSTEM/Q7DFINIT; DFBH _ /Q7DFCL1.	8	DEBUG
BAD FILENAME	File name must conform to file naming rules.	8	TCOPY
BAD FUNCTION CODE FOR GET/SEND MESSAGE	Notify a system analyst.	8	UPDATE
BAD LEVEL	Invalid security level.	-	Op System
BAD LIB FILE lfn THE WORD 'LIBRARY' NOT FOUND	Self-explanatory.	8	LOAD
BAD LOGON ERR	LOGON statement is in error.	-	Op System
BAD MINUS PAGE	Minus page of the task is not set up correctly.	-	Op System
BAD NAME NO NAME OR DUPLICATE IDENT	Correct UPDATE directive.	4	UPDATE
BAD ORIGIN ADDRESS FOR GROUP	An origin address must be on a small or large page boundary.	8	LOAD
BAD SUFFIX	Valid values are A, B, C, or D.	-	Op System
BAD USER/ACCOUNT	Invalid user number and account on a LOGON command.	-	Op System
BAD USER/SUFFIX	Invalid user number and account on a LOGON command.	-	Op System
BATCH INPUT FILE RECORD TOO BIG	Correct TCOPY directive.	8	TCOPY
BATCH INPUT FILE OPEN ERROR	Correct TCOPY directive.	8	TCOPY
BATCH PROCESSING TASK NOT INITIATED BECAUSE - reason	Self-explanatory.	-	CARDREDR

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
BATCH PROCESSOR RUNNING ON THAT SUFFIX	User tried to logon with a batch suffix.	-	Op System
BETA BUFFER LENGTH ERROR	Probable software failure.	8, 4	Pool Utilities
BOUND IMPLICIT MAP ANOMALY	Garbage in the minus page. Rebuild the minus page.	-	Op System
CALL OF INACTIVE OR PURGED COMDECK	Informative message.	4	UPDATE
CALL OF NON COMDECK	Change CALL to existing common deck name.	4	UPDATE
CALLER IS NOT A PRIVILEGED USER	Program has detected that the caller is attempting to illegally process files.	8	PF
CALLER NOT MEMBER OF poolname OR POOL NOT ATTACHED	Caller is not authorized to access pool or the pool is not attached.	-	PF
CANNOT ATTACH TO POOL	User not granted access or is not the pool boss.	8	PATTACH
lfn CANNOT BE OPENED	Self-explanatory.	8	DEBUG LOOK
CANNOT OPEN lfn	Self-explanatory.	8	DUMP
CANNOT RESTART - REGTBL full	Cannot restart a drop file. System table REGTBL is full. Try again later.	-	Op System
CANNOT SEND MESSAGE TO OPERATOR	Notify a system analyst.	8	TCOPY
CANT DESTROY EXISTING DROP FILE	Modified pages written to drop file and cannot be purged.	-	Op System
CENTRAL MEMORY PARITY ERROR	Rerun job.	-	Op System
CHANNEL FAILURE	Notify a system analyst.	8	TCOPY
CLOSE ERROR R=#rrrr SS=#ss ON FILE lfn	See CLOSE FILE system message for explanation of error codes.	-	Q7GETFIL
reason - COMMAND IGNORED		4	LOOK
COMPARE TERMINATED - END OF FILE lfn	Informative message.	-	COMPARE
COMPILE FILE TOO SMALL	Rerun with larger compile file	8	UPDATE
CONTRADICTION, LIST=O AND OUTPUT=	Correct control statement.	8	OLE
CONTROL CARD ERROR	Correct control statement syntax. Parameters should be separated from the control statement name by a (, / = + - or blank.	0	BATCHPRO
CONTROLLEE FORMAT ERROR	Error in format of the controllee option.	8	LOAD
CONTROLLEE FILE lfn IS TOO SMALL, NEED nnnnnnn SMALL PAGES	Rerun with a larger controllee file specified.	8	LOAD
CONTROLLER WAITING FOR MESSAGE	Probable software failure.	8	Pool Utilities
COPY COMPLETED	Informative message.	0	TCOPY

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
COPY TERMINATED BY EOF	Informative message.	0	TCOPY
COPY TERMINATED BY END OF TAPE	Informative message.	0	TCOPY
COPY TERMINATED BY SHORT RECORD	Informative message.	0	TCOPY
COULD NOT CREATE FILE=OUTPUT	File OUTPUT is already open.	8	DUMP
COULD NOT CREATE NEWPL	No mass storage space available. Rerun job.	8	UPDATE
COULD NOT LINK CORRECTION TO DIRECTORY ENTRY	Notify a system analyst.	4	UPDATE
CR STR TOO BIG	Character string is bigger than space allotted; too many characters in the input symbol.	8	LOAD
CREATE ERROR R=#rrrr SS=#ss ON FILE lfn	See CREATE FILE system message for explanation of error codes.	-	Q7GETFIL
CREATE PERMANENT FILE lfn	Informative message.	-	DEFINE
CREATION RUN ABORT	Informative message.	8	UPDATE
CREATE PERMANENT FILE-LFN	Informative message	0	DEFINE
DATA BASE EXCEEDS FIELD	Length of the data base is greater than FFFFFFFF words.	8	LOAD
DATA BEYOND FF	The user has done an EREG such that the data to be entered would go beyond entering in register FF. For example, EREG FE,1,2,3,4,5.	4	DEBUG
DATA EXCEEDS BUFFER	Correct TCOPY directive. (Check that the correct tape is mounted.)	8	TCOPY
DECKS OR IDENTS OUT OF ORDER	Correct range parameter.	4	UPDATE
DELIMITER COUNT OVER 200	Probable software failure.	8	Pool Utilities
DESTINATION USER PERM. FILE SPACE EXCEEDS INSTALLATION LIMIT	Installation-controlled disk space limit is exceeded at the new user.	8	GIVE
DESTROY ERROR R=#rrrr SS=#ss ON FILE lfn	See DESTROY FILE system message for explanation of error codes.	-	Q7GETFIL
DISC AT BEGINNING OF FILE	Informative message.	0	TCOPY
DISC PARITY ERR.	Notify a system analyst.	8	TCOPY
DISCFILE CREATED	Informative message.	0	TCOPY
DISCFILE OPENED	Informative message.	0	TCOPY
lfn DOES NOT EXIST	Self-explanatory.	8	DEBUG
DONE (last message)	Last message from TCOPY.	0	TCOPY
DROP FILE CONFLICT	Rerun job.	8	TCOPY
DROP FILE IOC DOES NOT VERIFY	Either the drop file does not exist or the ioc does not match the file index.	-	Op System

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
DROP FILE IS PRI-OPENED	Cannot restart a drop file that is open. It might be open due to a privileged DUMPF.	-	Op System
DROP FILE MAP FULL, PAGE NOT MAPPED	Reprogram job to use fewer drop file map entries at one time.	-	Op System
DROP FILE OVERFLOW	Rerun with a larger drop file.	-	Op System
DROP FILE OVERFLOW CAUSED BY CALL CALL TO VS	Rerun with a larger drop file.	-	Op System
DROP FILE SIZE NOT APPLICABLE FOR NON-CODE TYPE FILE	DROP parameter cannot be specified for data file.	8	SWITCH
DROP FILE TOO SMALL	New drop file will not hold existing drop file page.	-	Op System
DUMPING lfn	Informative message identifying file being dumped.	-	DUMPF
DUPLICATE POOL NAME	Pool name already exists.	8	PCREAT
DUPLICATE PUBLIC FILES	Informative message at the operator's console during LOGON.	-	Op System
EMPTY INPUT FILE	Correct input file.	8	UPDATE
ENCOUNTERED INVALID CHARACTER	Correct input card.	4	UPDATE
ENCOUNTERED READ ERRORS ON OLDPL (BAD DATA)	Notify a system analyst.	8	UPDATE
ENCOUNTERED UNPROCESSED MODIFICATIONS	Correct UPDATE directive.	4	UPDATE
END OF AREA IN DISCFILE	Informative message.	-	TCOPY
ERROR ENCOUNTERED WITH LIST FILE INDEX, SS=ss	See LIST FILE INDEX OR SYSTEM TABLE system message for explanation of error code.	8	PF
ERROR ENCOUNTERED WITH MAP MESSAGE, SS=nnn	Error in attempt to map in/out buffer areas (free space). See MAP system message for explanation of error code.	8	PF
ERROR ENCOUNTERED WITH yyyyyyyy MESSAGE -lfn, SS=ss	See system message CREATE, OPEN, CLOSE, DESTROY, ROUTE AND FILE DISPOSITION for explanation of error code.	8	PF
ERROR - EXPLICIT I/O FOR FILE lfn, yyyyyy=nnn	Program has encountered a cerr, serr, or status error for file lfn. This file is bypassed and processing continues for disk files.	4	PF
ERROR IN ATTEMPT TO DEQUEUE SERVICE STATION FILE	Notify a system analyst.	-	CARDREDR
ERROR IN ATTEMPT TO GIVE AN INPUT FILE TO ITS OWNER (FUNCTION = #0008, SS = #ss)	See system message GIVE FILE for explanation of error code.	-	CARDREDR
ERROR IN ATTEMPT TO INSTALL ROUTING INFORMATION	Notify a system analyst.	-	CARDREDR
ERROR IN ATTEMPT TO TRANSFER CARD INPUT FILE TO MASS STORAGE	Notify a system analyst.	-	CARDREDR

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
ERROR IN CHANGING ACCOUNT NUMBER OF INPUT FILE (FUNCTION = #0008, SS = #ss)	See system message CHANGE FILE NAME OR ACCOUNT for explanation of error code.	-	CARDREDR
ERROR IN CLOSING CENTRAL CARD INPUT FILE (FUNCTION = #0005, SS = #ss)	See system message CLOSE FILE for explanation of error code.	-	CARDREDR
ERROR IN CREATING CARD INPUT FILE (FUNCTION = #0001, SS = #ss)	See system message CREATE FILE for explanation of error code.	-	CARDREDR
ERROR IN CREATING CENTRAL FILE FOR CARD INPUT FILE AND ERROR IN ATTEMPT TO DESTROY SERVICE STATION FILE (FUNCTION = #0001, SS = #ss)	See system message CREATE FILE for explanation of error code.	-	CARDREDR
ERROR IN LIST FILE INDEX CALL (FUNCTION = #0009)	Notify a system analyst.	-	CARDREDR
ERROR IN Q7GETFIL	File creation, destroy, or opening problem.	8	OLE
ERROR IN Q7PROMPT	Notify a system analyst.	8	OLE
ERROR lfn DESTROY SS=ss	Public file cannot be removed from public file list. See DESTROY FILE system message for explanation of error code.	4	EDITPUB
ERROR lfn GIVE SS=ss	New file cannot be added to public file list. See GIVE FILE system message for explanation of error code.	8	EDITPUB
ERROR OCCURRED DURING MAP-IN OF I/O BUFFER SS = ss	Tape buffers for explicit I/O could not be mapped-in. The area reserved for buffers must not be mapped-in prior to calling the tape subroutines. See EXPLICIT I/O system message for explanation of error code.	8	Tape Sub
ERROR OCCURRED IN ATTEMPT TO DESTROY A FILE THAT COULD NOT BE GIVEN TO A USER (FUNCTION = #0002, SS = #ss)	See system message DESTROY FILE for explanation of error code.	-	CARDREDR
ERROR OCCURRED IN ATTEMPT TO DESTROY INPUT FILE ON SERVICE STATION DEVICE	Notify a system analyst.	-	CARDREDR
ERROR OCCURRED IN ATTEMPT TO READ INPUT FILE FROM SERVICE STATION DEVICE	Notify a system analyst.	-	CARDREDR
ERROR ON MAP-IN DROP FILE ENTRY	Notify a system analyst.	8	DEBUG
ERROR OPENING FILE lfn SS=ss	See OPEN FILE system message for explanation of error code.	8	LOAD
ERROR OPENING LIBRARY FILE lfn SS=ss	See OPEN FILE system message for explanation of error code.	8	LOAD
EXCEEDED MAXIMUM SEQUENCE NUMBER	Maximum sequence number is 65 535.	4	UPDATE
EXCEEDED SPECIFIED PN'S	List of pack names insufficient for file creation. Processing continues with system default PN.	-	PF

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
EXISTING FILE lfn OPENED, MAXIMUM LENGTH = #nnnn	Request specifies gmode=c and file lfn exists as an attached permanent file with a maximum length of nnnn small pages, which is less than the maximum length set by the user.	-	Q7GETFIL
EXISTING LOCAL FILE lfn MADE PERMANENT	Informative message.	-	DEFINE
EXPECTED/BAD NAME ENCOUNTERED	Notify a system analyst.	4	UPDATE
EXPECTED FILE NAME NOT ENCOUNTERED	Correct UPDATE directive.	4	UPDATE
EXPECTED IDENT NAME NOT FOUND	Correct UPDATE directive.	4	UPDATE
EXPECTED SEQUENCE NUMBER NOT FOUND	Correct UPDATE directive.	4	UPDATE
EXPLICIT I/O ERROR CERR=cerr SERR=serr	See EXPLICIT I/O system message for explanation of error codes.	8	TCOPY
EXPLICIT I/O ERROR - UNABLE TO OPEN BUFFERS	Self-explanatory.	8	COPY
EXPLICIT I/O (GIVE-UP) ERROR	Self-explanatory.	8	COPY
EXPLICIT I/O ERROR - UNABLE TO OPEN/CLOSE BUFFER	Self-explanatory.	8	COPY
EXPLICIT I/O ERROR - UNABLE TO READ filename	Self-explanatory.	8	COPY
EXPLICIT I/O ERROR - UNABLE TO WRITE filename	Self-explanatory.	8	COPY
FATAL SYSTEM ERROR	Rerun job.	-	Op System
FILE ALREADY EXISTS	Duplicate file names cannot exist for the same ownership category.	8	TCOPY REQUEST
FILE lfn ALREADY EXISTS	File name specified already exists in the file index for this user.	8	DEFINE
FILE lfn CURRENTLY OPENED	File not returned.	4	RETURN
FILE CLOSE ERR.	Notify a system analyst.	8	TCOPY
FILE CLOSED	Informative message.	-	TCOPY
FILE COPY BUG n (INTERNAL ERROR)	Notify a system analyst.	8	TCOPY
lfn FILE ERROR	Error in closing the controllee file.	8	LOAD
FILE HAS SAME NAME AS A PUBLIC FILE	Switch file name before giving it to another user.	4	GIVE
FILE INDEX FULL	Other users have to free space in the file index before you can attach your pool.	8	PATTACH
FILE INDEX FULL	Other users must destroy some of their files or logoff to free space in the file index. Relogon (reenter the LOGON command after having previously issued a BYE) to bring in private files.	8	TCOPY

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
FILE INDEX FULL. NONE OF YOUR PRIVATE FILES AVAILABLE.	Other users must destroy some of their files or logoff to free space in the file index. Relogon (reenter the LOGON command after having previously issued a BYE) to bring in private files.	-	Op System
FILE INDEX FULL. UNABLE TO CREATE FILE.	Notify a system analyst.	8	REQUEST
FILE INDEX IS FULL. UNABLE TO CREATE FILE lfn	Notify a system analyst.	8	DEFINE
FILE lfn ALREADY EXISTS	Attached file name already exists or is a duplicate permanent file name.	8	SWITCH
FILE lfn ATTACHED TO ANOTHER SUFFIX	File not available under suffix specified.	8	ATTACH
FILE lfn DOES NOT EXIST	File does not exist or is not attached.	8	SWITCH
FILE lfn DOES NOT EXIST	In batch mode, utility aborted.	8	ROUTE
FILE lfn IS ACTIVE	Cannot route a file that is currently active. ROUTE is aborted.	8	ROUTE
FILE lfn IS PRIVILEGED OPEN	Self-explanatory.	8	ATTACH
FILE lfn NOT GIVEN TO id BECAUSE	Subsequent message identifies cause of GIVE failure.	4	GIVE
FILE lfn TOO BIG FOR SERVICE STATION	File larger than #1000 blocks.	8	TOAS ROUTE
FILE IS A SOURCE OR DROP FILE	File owner not changed.	4	GIVE
FILE IS STILL ACTIVE	File owner cannot change while file is open.	4	GIVE
FILE MARK WRITTEN	Informative message.	-	TCOPY
FILE NAME ALREADY EXISTS IN RECEIVERS FILE INDEX	File remains with old owner.	4	GIVE
FILE NAME lfn IN USE AS A LOCAL FILE	File name already used.	8	ATTACH
FILE NOT AVAILABLE	File name not in the file index.	8	TCOPY
FILE OPEN, DENSITY COULD NOT BE CHANGED	Informative message.	-	TCOPY
FILE PROTECTED TAPE	Attempted to write to a tape without a write ring.	8	TCOPY
FILE SEGMENT TABLE IS FULL	Rerun job.	-	Op System
FILES COMPARED EQUALLY	Informative message.	0	COMPARE
FIRST FILENAME ILLEGAL - NO FILES PURGED	Self-explanatory.	8	PURGE
FOLLOWING CARDS ARE SKIPPED - NOT IN INSERT MODE	Informative message.	4	UPDATE
FORMAT ERROR	Correct control statement.	8	DEBUG EDITPUB

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
FOUND LOGICAL RECORD TOO LONG (BLOCKBCD MODE)	Maximum record length is #3FF characters. Correct TCOPY directive.	8	TCOPY
GENERATION OF UNIQUE IDNAME FAILED	Notify a system analyst.	4	UPDATE
GET BUFFER ERROR R=XX (FC)=XX	Error in F004 call. Return code and contents of register FC are given.	8	SAVEPF
GET MESSAGE IS FROM LEVEL 1	Probable software failure.	8	Pool Utilities
GO	Operator response to tape condition. Operation continues by ignoring previously reported condition, or continues normally because the operator corrected the condition.	-	Tape Sub
GROUP ORIGIN AT ADDRESS WHICH IS ALREADY ALLOCATED	An attempt was made to allocate a group of modules or common blocks where another module or common block is already allocated.	8	LOAD
IDENT DOES NOT EXIST	Correct UPDATE directive.	4	UPDATE
ILLEGAL BKPT	DEBUG has obtained control from an unexpected point in the controllee, namely, from a point at which DEBUG has not set a breakpoint.	4	DEBUG
ILLEGAL CHAR NUMBER	Correct hexadecimal number to include only digits 0 through 9 and letters A through F.	8	LOAD
ILLEGAL COMMAND	Illegal input parameters.	8	LOAD
ILLEGAL C504 REQUEST	User jobs cannot issue a C504 request.	-	Op System
ILLEGAL DATE CORRECTED TO mmddyy	DT parameter is not a legal date; it is changed to the nearest legal date as shown.	-	PF
ILLEGAL DENSITY	Tape density parameter must be 2, 5, 8, or 16.	8	TCOPY
ILLEGAL DEVICE NUMBER	Self-explanatory.	8	TCOPY
ILLEGAL FIRST CARD	STORE card required.	-	BATCHPRO
ILLEGAL INSTRUCTION	Illegal instruction at address given.	-	Op System
ILLEGAL PARAMETER	Self-explanatory.	8	OLE
ILLEGAL REQUEST	Illegal Alpha function code.	-	Op System
INADR EXCEEDS infile FILE LENGTH	I parameter is greater than the number of words in infile.	8	COPY
IN FILE CHARACTERS NOT VALID ON OUT FILE	Correct TCOPY directive.	8	TCOPY
INPUT FILE TOO BIG	Input file of CYBER 200 job submitted at CYBER front-end was too big.	-	CYBERIN
INT DATA ** MODE = 1 TYPE ILLEGAL	Mode 1 of an interpretive data type is illegal - probably the most common with type 9 mode 1. Table type 101.	8	LOAD
INT REL ** MODE = 2 TYPE DOES NOT EXIST	Mode 2 of an interpretive data type does not exist. Table type 201.	8	LOAD

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
INVALID BLOCK SIZE ON OUTPUT	Correct TCOPY directive.	8	TCOPY
INVALID CARD	Correct input.	4	UPDATE
INVALID DIRECTIVE FOR CREATION RUN	Only READ, DECK, COMDECK, and ADDFILE can appear in a creation run.	4	UPDATE
INVALID EXPRESSION: expression ENTER REPLACEMENT OR "CANCEL"	Enter correct expression, enter CANCEL to abort, or hit NEW-LINE key to ignore bad expression.	8	Q7KEYWRD
INVALID FILE NAME	File names must be 1 through 8 letters or digits beginning with a letter.	4	UPDATE
INVALID FILE OR FUNCTION	Correct UPDATE input.	8	UPDATE
INVALID FILE NUMBER OR FUNCTION	Correct UPDATE input.	4	UPDATE
INVALID OPTION	Error in system message for pool file manager.	8, 4	Pool Utilities
INVALID POOL NAME	Pool name is greater than eight characters long or does not begin with a letter.	8, 4	Pool Utilities
INVALID SEQUENCE NUMBER	Correct UPDATE directive.	4	UPDATE
INVALID UPDATE CHECK WORD	Notify a system analyst.	8	UPDATE
INVALID USER NUMBER	User tried to use a reserved number.	-	Op System
I/O CONNECTOR IN USE	Self-explanatory.	8	TCOPY
I/O ERROR RECEIVED BY WRPLY	Rerun job.	-	Op System
IOC DOESN'T VERIFY	Invalid IOCs in a drop file being restarted. Usually means there is an IOC for a file which is no longer available.	-	Op System
IOC FOR lfn ALREADY IN USE	Self-explanatory.	8	DEBUG
ITEM COUNT OUT OF RANGE 0,4096	Probable software failure.	8	Pool Utilities
JOB ABORTED	Informative message.	-	CARDREDR
JOB CARD ERROR	Correct job statement.	-	BATCHPRO
JOB FILE VACUOUS	The first record of a batch job must contain ASCII character control statements.	-	BATCHPRO
JOB MAY NOT USE A TAPE	Notify a system analyst.	8	TCOPY
KILL	Operator response to tape condition. Tape operation is aborted.	8	Tape Sub
LARGE PAGE LIMIT EXCEEDED	An instruction in the program requires more large pages than that job category allows before it can execute.	-	Op System
LENGTHS DON'T MATCH FOR COMMON BLOCK BLOCKNAME	Self-explanatory.	4	LOAD
LIBRARY DIRECTORY AND INDEX TABLE FULL	Notify a system analyst.	8	OLE

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
LINK I/O ERROR (FC)=XX	Error in explicit I/O other than EOF or operator abort. Contents of register FC are given.	8	SAVEPF
LINKED MAINFRAME FILE ERROR	File error on CYBER or aborted by CYBER 200 operator. Check parameters and try again. (CYBER 200 Link Station only)	8	GETPF SAVEPF PURGE
LIST DISK STATUS TABLE ERROR R=XXXX	Self-explanatory.	8	SAVEPF
LIST FILE INDEX ERROR R=XXXX	Self-explanatory.	8	SAVEPF
LIST FILEI ERROR R=#rrr	See LIST FILE INDEX system message for an explanation of the error code.	8	RETURN
LIST RANGE ERROR R=XX (FC)=XX	Error in FO04 call. Return code and contents of register FC given.	8	SAVEPF
LOADING lfn	Informative message identifying file being loaded.	-	LOADPF
LOADMAP FORMAT ERROR	Bad input parameter for creating an output file.	8	LOAD
LOST DATA	Tape station error. Notify a system analyst.	8	TCOPY
MAGNETIC TAPE IS NOT AVAILABLE, CANNOT CONTINUE	Self-explanatory.	8	PF
MASS STORAGE ERROR	Notify a system analyst.	8	TCOPY
MASS STORAGE POSITIONING ERROR	Notify a system analyst.	8	TCOPY
MASTER CONTROL WORD DOES NOT MATCH OLD PL	Notify a system analyst.	4	UPDATE
MASTER DIRECTORY FILE xxxxxxxx IS FULL	The master directory, xxxxxxxx, containing the pseudo file names for the user's dumped files on disk is full. Remaining files are dumped to the next specified PN (packid).	4	PF
MAXIMUM ERRORS EXCEEDED	More than 256 errors. Correct and rerun.	4	UPDATE
MAXIMUM INPUT RECORD LENGTH EXCEEDED	Check input file structure.	8	UPDATE
MAXIMUM NUMBER OF CORRECTION HISTORY BYTES REACHED	Create new program library.	4	UPDATE
MAXIMUM NUMBER OF FIELDS EXCEEDED	Correct control statement.	4	UPDATE
MESSAGE ERROR	Error in system GET A MESSAGE call.	8	LOAD
MESSAGE FROM CONTROLLER READY	Probable software failure.	8	Pool Utilities
MISSING INPUT FILE	Correct control statement.	8	OLE
MODULE LIMIT ON OBJECT FILE EXCEEDED	Notify a system analyst.	8	OLE
MODULE ON FILE lfn HAD A MODULE HEADER LENGTH OF ZERO	Bad module header format in the input file.	8	OLE

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
MODULE ON FILE lfn HAS NO HEADER	Module does not have a header table.	8	OLE
MORE THAN ONE ALTERNATE FILE WAS ATTEMPTED	Correct UPDATE directive.	8	UPDATE
MORE THAN ONE OMIT FOR FILE lfn	Correct control statement.	8	OLE
MTuu message	See corresponding NTuu message.		
MTuu vsn NOT AVAILABLE, CONTINUE WITH NEXT VSN	Specified vsn was not available; the program continues with the next specified vsn.	4	PF
MTuu vsn NOT AVAILABLE, CANNOT CONTINUE	Specified vsn is not available, and the program cannot continue with subsequent vsn's.	8	PF
NAME ALREADY EXISTS IN DIRECTORY	Change duplicate deck name.	4	UPDATE
NAME DOES NOT EXIST IN DIRECTORY	Change deck name or directive requested.	4	UPDATE
NEW VOL WAS NOT ASSIGNED	Labeled tape file could not be closed because the volume was at end-of-tape. Assign additional volume using Q8ASGNTP.	8	Tape Sub
NO aaa ACCESS ON FILE lfn	Request specifies either gmode = U and file lfn exists but does not have write access and the file access cannot be changed, or gmode = O but the file lfn does not have read access and the file access cannot be changed. The file access cannot be changed if the job mode is batch, or if an interactive user responds with an N to the request: MAY aaa ACCESS BE ADDED TO FILE lfn, Y OR N?	-	Q7GETFIL
NO ALPHA POINTER	Virtual system was called for an Alpha message, but the pointer to the Alpha message was O.	-	Op System
NO DISC SPACE AVAILABLE TO CREATE FILE	Insufficient disc space to create file. If the PACK option was used, try creating the file on another pack by eliminating the PACK keyword or specifying another packid.	8	REQUEST
NO DISC SPACE AVAILABLE TO CREATE FILE lfn	Insufficient disc space to create file. If the PACK option was used, try creating the file on another pack by eliminating the PACK keyword or specifying another packid.	8	DEFINE
NO DISC SPACE FOR EXTENSION ON FILE	Rerun job with this file on a disk that has enough space for this file.	-	Op System
NO DISPOSITION CODE FOR FILE lfn	Set the dc option.	8	ROUTE
NO DROP FILE	There is no drop file for this task.	-	Op System
NO ENTRY	No entry point found.	8	LOAD
NO ERROR EXIT ADDRESS	Task issued an Alpha/Beta system call which returned an error, but the error exit address field in the Alpha portion was O.	-	Op System
NO EXT/ENT POINTER FILE lfn	No type 2 table pointer found.	8	LOAD

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
NO FILE	Either the file name does not exist, or the number of characters in filename was not 1 through 8.	-	Op System
NO FILE NAME FOR READ	Add file name to READ directive.	4	UPDATE
NO FILE SEGMENT TABLE SPACE	System file segment table full. Rerun.	-	Op System
NO FILES EXIST TO LIST. UTILITY TERMINATED	Self-explanatory.	8	FILES
NO FILES TO xxxxxx	Informative message.	0	PF
NO FST ORDINAL WITH C50X REQUEST	Rerun with an FST ordinal.	-	Op System
NO LIBRARY DIRECTORY SEARCH	Job has an address in library space. Library space cannot be used at this time.	-	Op System
NO LOGONS	Interactive terminal logon lines are inhibited.	-	Op System
NO MASS STORAGE SPACE AVAILABLE	Rerun job.	8	TCOPY
NO MESSAGE POINTER FOLLOWS EXIT FORCE	Rerun with a correct exit force.	-	Op System
NO MESSAGE READY	Probable software failure.	8	Pool Utilities
NO MORE SEGMENT SPACE IN FILE1, FILE lfn	File has been extended three times. Rerun with a larger file.	-	Op System
NO PARAMS GIVEN	Correct DEBUG control statement.	8	DEBUG
NO PP	No task in execution to break.	-	Op System
NO SOURCE FILE	There is no controllee file.	-	Op System
NO SUCH ATTACHED POOL EXISTS. UTILITY TERMINATED.	Self-explanatory.	8	FILES
NO SUCH ATTACHED FILE EXISTS	File is not attached.	4	GIVE
NO SUCH MODULE	Value entered for name=location was not the name of a module in the controllee.	4	DEBUG
NO SUCH SYMBOL	Symbol specified was not found as a symbol of current type (S or L) in current module.	4	DEBUG
NO SWITCH PARAMETERS FOUND, FILE NOT ALTERED.	File name was only parameter found.	4	SWITCH
NO TIME IN BANK	Time in repository bank is reduced to zero.	-	Op System
NO TIME LEFT FOR drop file lfn	Rerun with a larger time limit.	-	Op System
NO TL	Zero time limit (TL) specified.	-	Op System
NO WRITE PERMISSION - COMMAND IGNORED	Self-explanatory.	-	LOOK
NON-EXECUTABLE FILE	File requested is not a virtual code file (file type other than 2).	-	Op System
NON-EXISTENT FILE lfn	Specified file is not a local or attached permanent file.	4	RETURN PURGE

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
NON-MATCHING WORDS	Listed words did not compare equally.	4	COMPARE
NOT A LOGON	First word of the LOGON command must be LOGON.	-	Op System
NOT A USER	Invalid user number.	-	Op System
NOT ALL FILES ATTACHED	Some permanent files could not be attached when the * option was specified.	8	ATTACH
NOT ENOUGH TIME FOR THIS JOB	Time limit specified exceeds time remaining in repository bank.	-	Op System
NOT EXPECTING SEQUENCE NUMBER	Correct UPDATE directive.	4	UPDATE
NTuu ASSIGNED - tape lfn	Informative message.	-	Tape Sub
NTuu ATTEMPT TO WRITE ON UNEXPIRED FILE	Tape mounted for a tape write operation contains an expiration date that is still within the retention period. Operator response can be GO or KILL.	-	Tape Sub
NTuu BAD OPERATOR RESPONSE	Operator entered invalid response to a tape subroutine message. The original message is reissued.	-	Tape Sub
NTuu BLOCK COUNT ERROR (nnnnnn)	Number of blocks read from a labeled input tape does not match the block count written in the trailer label. nnnnnn is the difference from the block count specified by the trailer label.	-	Tape Sub
NTuu BLOCKS READ = nnnnnn	Informative message.	-	Tape Sub
NTuu BLOCKS WRITTEN = nnnnnn	Informative message.	-	Tape Sub
NTuu COULD NOT READ TRAILER LABEL	Hardware error while attempting to read a tape trailer label.	8	Tape Sub
NTuu ENTER VSN	Tape subroutines require additional verification of vsn when the tape volume is unlabeled. Operator must respond with a 1 through 6 character vsn in a TELL DB command.	-	Tape Sub
NTuu EOT, WRITE ABORTED	Attempt to write data has been made after the end-of-tape reflector sensed.	8	Tape Sub
NTuu EOT, READ ABORTED	Attempt to read data has been made after an EOV1 label sensed. A new volume must be assigned with Q8SGNTP when an end-of-volume is sensed.	8	Tape Sub
NTuu INVALID HEADER LABEL	Valid HDR1 label could not be found for a labeled file.	8	Tape Sub
NTuu LOST DATA - PRU = nnnnnn	Hardware error occurred during a data transfer and specified physical tape record has been lost. If the task has requested return of control after data transfer errors, an error code is returned; otherwise, the task is aborted.	8	Tape Sub
NTuu MISSING HDR1 LABEL	File header label not present following the VOL1 of a labeled input tape.	8	Tape Sub

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
NTuu NO EOF1 LABEL	No trailer label present for a labeled input file.	8	Tape Sub
NTuu NO WRITE ENABLE	The tape mounted for a tape write operation does not contain a write enable ring. Operator response can be GO, RECHECK, or KILL.	-	Tape Sub
NTuu OPERATOR HAS ABORTED TAPE REQUEST	Operator aborted tape request during label processing.	8	Tape Sub
NTuu RING NOT ALLOWED	Tape mounted with a ring for a tape read operation.	-	Tape Sub
NTuu RPE UNRVD, CANNOT READ 1ST RECORD/VOL1	Fatal parity error occurred when opening a tape input volume.	8	Tape Sub
NTuu RPE UNRVD, CANNOT READ TRAILER LABEL	Fatal parity error occurred when closing a labeled input file.	8	Tape Sub
NTuu RPE UNRVD, CANNOT VERIFY SCRATCH TAPE	Fatal parity error occurred when checking an intended output tape volume for unexpired data files. Operator response can be GO, RECHECK, or KILL.	-	Tape Sub
NTuu RPE UNRVD - PRU = nnnnnn	Fatal parity error occurred during a read data request. If the task has requested return of control after data transfer errors, an error code is returned; otherwise, the task is aborted.	8	Tape Sub
NTuu TAPE HARDWARE ERROR, subroutine CANNOT CONTINUE	Fatal hardware error sensed from which the specified tape subroutine cannot recover.	8	Tape Sub
NTuu UNEXPIRED FILES	Tape mounted for a tape write operation contains an expiration date that is still within the retention period. Operator response can be GO, RECHECK, or KILL.	-	Tape Sub
NTuu UNIT NOT READY	Unit assigned for a tape request is not ready. Operator response can be GO, RECHECK, or KILL.	-	Tape Sub
NTuu UNLABELED	No label on a tape specified as labeled. Operator response can be GO, RECHECK, or KILL.	-	Tape Sub
NTuu WPE UNRVD - PRU = nnnnnn	Fatal write parity error detected for specified physical tape record.	8	Tape Sub
NTuu WPE UNRVD, CANNOT WRITE EOF	Fatal write parity error detected in the trailer label of a labeled output file.	-	Tape Sub
NTuu WRONG DEVICE	Device assigned by the operator is different from the device requested by the user. A request for reassignment of the tape follows.	-	Tape Sub
NTuu WRONG VSN	Volume serial number does not match that specified by the user. Operator response can be GO, RECHECK, or KILL.	-	Tape Sub
OLDPL DIRECTORY CANNOT BE PROCESSED	Notify a system analyst.	8	UPDATE
OLE TERMINATED	Informative message.	-	OLE

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
OMIT FILE lfn NOT AN INPUT FILE	Correct control statement.	8	OLE
OMIT MODULE modname NOT ON FILE lfn	The module does not exist in the input file.	8	OLE
OMIT PARAMETER MISSING A FILE OR MODULE	Correct control statement.	8	OLE
ON TTY xxxx	User numbers cannot logon to more than one terminal at a time.	-	Op System
OP COMPLET - END OF TAPE	Informative message.	-	TCOPY
OPEN ERROR R=#rrrr SS=#ss ON FILE lfn	See OPEN FILE system message for explanation of error codes.	-	Q7GETFIL
OPERATOR INITIATED TAPE ERROR	Notify a system analyst.	8	TCOPY
OPERATOR NO.=uuuuuu	The system operator attempted to logon under an invalid user number. The correct operator number is uuuuuu.	-	Op System
OPERATOR NOT RUNNING OR IOC NOT AVAILABLE	Ioc is not available for assignment of the requested tape. Possibly too many files open or OPERATOR not logged on.	8	Tape Sub
** OPERATOR REQUESTED ABORT **	Self-explanatory.	8	TCOPY
ORIGIN NOT ON LARGE PAGE BOUNDARY - ORIGIN = neworigin	Warning - GRLPALL option.	4	LOAD
OUT OF BOUND MEMORY REFERENCE	Attempt to access space outside the task virtual space.	-	Op System
OUTPUT FILE IS IMPROPERLY NAMED	Correct GIVE control statement.	4	GIVE
OVERDRAWN	Insufficient time remains in repository bank; results from (sc)G status request.	-	Op System
OVERLAP IN BI MAP FOR lfn	Bound implicit overlap error occurred during open.	8	DEBUG
OUTADR EXCEEDS outfile FILE LENGTH	O parameter is greater than the number of words in outfile.	8	COPY
packid PACK NOT AVAILABLE	Specified disk pack was not available; the program continues with the next specified PN (packid).	4	PF
PAGE SIZE CONFLICT IN DROP FILE	Small page fault but the drop file is in large pages, or vice versa.	-	Op System
PARAMETER ERROR	Correct TCOPY directive.	8	TCOPY
PARAMETER ERROR - FILE lfn	Value passed to GETFILE in the parameter table is illegal.	-	Q7GETFIL
PARAMETER OR FORMAT ERROR	Error in parameter specifications.	8	Pool Utilities PF
PARAMETER OR FORMAT ERROR FOUND. UTILITY TERMINATED	Correct FILES control statement.	8	FILES
PARAMETER OR FORMAT ERROR FOUND. UTILITY TERMINATED	Correct GIVE control statement. File remains with old owner.	8	GIVE

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
PARAMETER TOO LARGE	Correct TCOPY directive.	8	TCOPY
POOL ACCESS DIRECTORY FULL	Limit of 32 user identifiers per pool in the access directory.	8	PACCESS
POOL ALREADY ATTACHED	Attempt to reattach to pool.	8	PATTACH
POOL FILE lfn CURRENTLY OPENED	Self-explanatory.	4	PURGE
POOL LIST FULL	Unable to create a pool; limit of 512 file pools has been reached.	8	PACCESS
POOL NAME IS ILLEGAL OR GIVER IS NOT A MEMBER OF THIS POOL	File remains with old owner.	4	GIVE
POOL NOT ATTACHED	User not attached to pool.	4	PDETACH
POOL UTILITIES UNAVAILABLE	Files PAD and/or PLIST missing. Notify a system analyst.	8	Pool Utilities
PROBLEM WITH FILE	Notify a system analyst.	8	UPDATE
PROCESSED INVALID DIRECTIVE IN ALTERNATE FILE	Correct directive.	4	UPDATE
PUBFILE lfn DELETE ERROR ss	A public file cannot be removed from the public file list. See DESTROY FILE system message for explanation of error code.	4	EDITPUB
PURGE ERROR R=#rrr SS=#ss ON FILE lfn	See DESTROY system message for an explanation of the error code.	8	PURGE
QUALIFYING PARAMETER NOT PERMITTED WITHIN A LIST	Remove qualifying parameter.	8	DEFINE
RANGE IS NOT PERMITTED FOR YANK DECK	Correct YANKDECK directive.	4	UPDATE
READ ERROR, CALL=XX R=XX (FC)=XX	Error in C500 call. Call identifier, return code, and contents of register FC are given.	8	SAVEPF
RECEIVER OF FILE IS NOT ALLOWED TO ACCESS IT	File remains with old owner.	4	GIVE
RECHECK	Operator response to tape condition. A request is made for reassignment of an output tape volume; the operator can assign a tape mounted on a different unit.	-	Tape Sub
RECURSIVE CALL NOT PERMITTED	Correct CALL references.	4	UPDATE
REQUEST TO PURGE FILE IS NOT FROM POOL BOSS.	Self-explanatory.	4	PURGE
REQUESTED PACK COULD NOT BE FOUND	The pack specified on the PACK option does not exist.	8	DEFINE REQUEST
REQUIRED PARAMETER MISSING. NEXT EXPRESSION IS: expression ENTER PARAMETER OR "CANCEL"	Required positional parameter missing. Given expression appeared in the position where a required parameter was expected.	8	Q7KEYWRD
RETURN ERROR R=#rrr SS=#ss ON FILE lfn	See DESTROY system message for an explanation of the error code.	8	RETURN

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
ROUTE ERROR R=#rrrr SS=#ss ON FILE lfn	See corresponding system message for an explanation of error codes.	-	Q7GETFIL
ROUTE ERROR ss	See ROUTE AND FILE DISPOSITION system message for an explanation of error code.	8	ROUTE
** RUN ABORTED **	Informative message.	8	TCOPY
SAY AGAIN	Special character (sc) is not followed by valid system inquiry character.	-	Op System
SBU MEMORY PARITY ERROR	Parity error occurred on read or write.	-	Op System
SECURITY LEVEL TOO HIGH	Invalid security level for this user number.	-	Op System
SECURITY VIOLATION	Self-explanatory.	-	TCOPY
SEND AGAIN	The state at this DB entry is zero; or job class is priority and privileged job permission flag is zero; or job is currently in interrupt mode, explicit I/O interrupt has occurred.	-	Op System
SEQUENCE NUMBER EXCEEDED	Create two decks if text cards exceed 65535.	4	UPDATE
SEQUENCE NUMBER NOT FOUND	Specified sequence number non-existent. Correct.	4	UPDATE
SKIP COMPLETED	Informative message.	-	TCOPY
SKIP TERMINATED BY EOP	Informative message.	-	TCOPY
SKIP # TOO BIG	Correct TCOPY directive.	8	TCOPY
SOURCE OR DROP FILE ANOMALY	IOC in bound implicit map is not 16 (source); or bound implicit map entry is outside of file bounds; or drop file (free space) map address overlap occurred.	-	Op System
SOURCE FILE TOO SMALL	Increase size of source file and rerun.	8	UPDATE
*** SRM ABORT - A GENFIT OR GENFITX WAS NOT ISSUED FOR LFN	Self-explanatory.	8	SRM
STANDBY JOB IS NOT ALLOWED TAPE ACCESS	Standby job cannot access tapes. Job is aborted.	8	Tape Sub
STATION lid NOT LOGGED IN	Invalid logical ID or station is not logged in for ST option. ROUTE aborted.	8	ROUTE
SYSTEM DROP FILE CREATE ERROR	Either the disk or file index is full.	-	Op System
SYSTEM ERROR IN DETERMINING SUFFIX. FC=xx,R=rrr	See system message indicated by function code for explanation of error code.	8	FILES
SYSTEM ERROR IN GET MESSAGE FROM TERMINAL. FC=xx,R=rrr	See system message indicated by function code for explanation of error code.	8	FILES
SYSTEM ERROR IN SEND MESSAGE TO TERMINAL. FC=xx,R=rrr	See system message indicated by function code for explanation of error code.	8	FILES
SYSTEM MESSAGE ERROR	Batch processor detected a system message error. Contents of Alpha and Beta words follow.	-	BATCHPRO

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
SYSTEM TABLES FULL, TRY AGAIN	The XEQ buffer table is full as more than 8 execute lines have been entered; or no DB entry can be obtained; or no user table entries are available.	-	Op System
TABLE TYPE NOT IMPLEMENTED	Refers to compiler output for the loader.	8	LOAD
TAPE FILE OPEN	Informative message.	-	TCOPY
TAPE PARITY ERR.	Rerun job.	8	TCOPY
TAPE RECORD EXCEEDS BUFFER SIZE	Correct TCOPY directive.	8	TCOPY
TAPEFIL REWOUND	Informative message.	-	TCOPY
THERE EXISTS MORE FILES THAN CAN BE PROCESSED	In issuing the system message LIST FILE INDEX for private user files, the area to contain the file entries is not large enough. Subsequently, DUMPF processes only the files received in the area. In order to obtain the remaining files, the number of files must be reduced (for example: P option or destroy some files).	0	PF
THIS USER NUMBER MAY NOT ALTER A PUBLIC FILE	Self-explanatory.	8	SWITCH
TOO MANY ERRORS	Correct UPDATE directives.	8	UPDATE
TOO MANY FILES OPEN	No more than 14 files can be open for batch execution or 16 files for interactive execution.	8	TCOPY
TRANSMISSION PARITY ERROR	Parity error occurred on read or write.	-	Op System
TROUBLE WRITING PERMANENT FILE INDEX FOR FILE lfn	Error occurred while updating the permanent file index. ROUTE aborted; try again.	8	ROUTE
TRY AGAIN	ROLL or BACK was given before a DISPLAY or ENTRY command and DEBUG has no point of reference for ROLL/BACK. CONTINUE was given before EXECUTE. A second EXECUTE command is given. STEP is given before EXECUTE.	4	DEBUG
UNABLE TO CLOSE FILE lfn	Notify a system analyst.	8	OLE
UNABLE TO CLOSE SERVICE STATION INPUT FILE	Notify a system analyst.	-	CARDREDR
UNABLE TO COMPLETE UTILITY. TRY AGAIN	Self-explanatory.	8	PURGE GIVE
UNABLE TO CREATE OUTPUT FILE. FC=xx,SS=ss	FILES received the indicated ss code on a CREATE. See system message indicated by function code for explanation of error code.	8	FILES
UNABLE TO CREATE OUTPUT FILE. FILE INDEX FULL. FC=xx,SS=ss	See system message indicated by function code for explanation of error code.	8	FILES
UNABLE TO DESTROY POOL	Users attached, files are in the pool, or user is not the pool boss.	4	PDESTROY
UNABLE TO ENLARGE DROP FILE	An error occurred during an attempt to CREATE a larger drop file.	8	DEBUG

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
UNABLE TO ENLARGE DROP FILE. TRY AGAIN	Self-explanatory.	8	PURGE
UNABLE TO FIND EXECUTE FILE	Self-explanatory.	-	Op System
UNABLE TO FIND EXTERNAL-ENTRY TABLE OF MODULE modname ON FILE lfn	Bad module structure.	8	OLE
UNABLE TO GET TIME AND DATE	Notify a system analyst.	8	OLE
UNABLE TO INITIALIZE UTILITY COPY	Error occurred when attempting to initialize the controllee copy to perform the SAVE option. ROUTE aborted.	8	ROUTE
UNABLE TO LIST FILE INDEX FOR lfn	Self-explanatory.	8	COPY
UNABLE TO LIST FILE INDEX. TRY AGAIN	File remains with old owner.	8	GIVE
UNABLE TO LIST FILE INDEX. TRY AGAIN	Self-explanatory.	8	FILES PURGE
UNABLE TO MAP-IN USER MINUS PAGE	Self-explanatory.	8	DEBUG
UNABLE TO MAP-IN USER FILE	Self-explanatory	8	DEBUG
UNABLE TO OPEN BUFFERS	Error occurred in an attempt to open buffers for explicit input/output.	8	COMPARE
UNABLE TO OPEN FILE	Error occurred in an attempt to open file for explicit input/output.	8	COMPARE
UNABLE TO OPEN FILE lfn	File not in file index.	8	OLE
UNABLE TO OPEN lfn	Error occurred in an attempt to open file for explicit I/O.	8	COPY
UNABLE TO OPEN FILE LOCATED ON SERVICE STATION DEVICE	Self-explanatory.	-	CARDREDR
UNABLE TO READ lfn	0 value not on block boundary and output file does not have read access.	8	COPY
UNABLE TO READ FILE	Error occurred during an explicit read of file.	8	COMPARE
UNABLE TO SAVE FILE	Error while copying file. ROUTE aborted.	8	ROUTE
UNABLE TO SEND MESSAGE	Error occurred in an attempt to send a message to the controller.	8	COPY
UNABLE TO UNLOCK CORE PREVIOUSLY LOCKED DOWN FOR I/O FILE TRANSFER	Self-explanatory.	-	CARDREDR
UNAVAILABLE MASS STORAGE ON SYSTEM PACK(S)	No disk space to create a file.	8	PF
UNDEFINED ERROR	Probable software error.	8, 4	Pool Utilities
UNDEFINED FILE NAME lfn	File name specified is nonexistent.	8	ATTACH
UNDEFINED NAME OR ALREADY IN GROUP	Grouping not done because of an undefined name or the element to be grouped is already in another group.	8	LOAD

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
UNDEFINED POOL NAME	Pool name specified is nonexistent.	8, 4	PACCESS PATTACH Pool utilities
UNDEFINED USER NUMBER	User identifier is not in legal range or not in the access directory.	8, 4	PACCESS PDELETE
** UNEXPECTED PROGRAM ERROR **	Program encountered unexpected error. Contents of the FIT are displayed, which indicates the cause of the fatal error.	8	PF
UNEXPECTED RESPONSE (ss= nn) FROM ATTACH MESSAGE FOR lfn	See ATTACH FILE system message for explanation of error code.	8	ATTACH
UNEXPECTED RESPONSE FROM OPEN, CLOSE, OR CHANGE MESSAGE. SWITCH UTILITY ABORTED.	Self-explanatory.	8	SWITCH
UNIT NOT READY	Self-explanatory.	8	TCOPY
UNLOCK PAGES ERROR R=XX (FC)=XX	Error in F010 call. Return code and contents of register FC are given.	8	SAVEPF
UNRECOVERABLE READ ERRORS FOR FILE lfn	Program has encountered unrecoverable read errors. File is bypassed and processing continues with the next file.	4	PF
UPDATE OBTAINED BAD DATA IN PROCESSING THIS CARD	Correct card.	4	UPDATE
USER PERMANENT FILE SPACE EXCEEDS INSTALLATION LIMIT	Self-explanatory.	8	DEFINE
USER SPECIFIED IS A PUBLIC LIST	Only EDITPUB execution under a privileged user number can establish public file ownership.	4	GIVE
VARIABLE RATES NOT DEFINED AT THIS INSTALLATION	Control statement contained VRI parameter and system installation parameter IP _ F _ VR is set to zero.	8	EDITPUB
WARNING *** ATTACHED POOLS	Informative message during LOGON. The user has attached pools.	-	Op System
WARNING *** CHECKPOINTED JOBS UNDER SUFFIX D jobname 1, jobname 2, jobname 3, jobname 4, jobname 5, ...	A logon under the checkpointed batch suffix D will abort the checkpointed jobs for this user. All checkpointed batch input files, checkpointed output files, and batch local files are destroyed. Permanent files attached to the checkpointed suffix are returned.	-	Op System
WARNING * DUPLICATE FILES	Informative message during LOGON. The user has duplicate files.	-	Op System
WARNING MODULE name FROM FILE lfn IS INACCESSIBLE AND THEREFORE NOT LOADED	Module is deleted from controllee since it cannot be referenced.	4	LOAD
WARNING MULTIPLE TRANSFER SYMBOLS DEFINED	More than one program entry point is defined.	4	LOAD
WARNING - NO OBJECT FILE CREATED	Informative message.	4	OLE
WARNING - OBJECT FILE LENGTH INCREASED TO nnnn PAGES	Informative message.	4	OLE

TABLE B-3. DIAGNOSTIC MESSAGES (Contd)

Message	Significance/Action	Return Code	Issued By
WARNING - OUTPUT FILE TRUNCATED	Make the output file size larger.	4	OLE
WARNING - UNABLE TO REDUCE OUTPUT FILE	Informative message.	4	OLE
WARNING UNSATISFIED EXTERNAL(S) DETECTED DURING LOAD	Routine referenced but not provided.	4	LOAD
WARNING xxxxxxxx IS DUPLICATE ENTRY POINT IN MODULES yyyyyyyy AND zzzzzzzz	xxxxxxxx is the entry name, and yyyyyyyy and zzzzzzzz are the module names. References to xxxxxxxx link to the entry in yyyyyyyy, which was the first encountered.	4	LOAD
WRITE ON READ-ONLY FILE	Job attempted to write to temporary space for which the job does not have write access.	-	Op System
WRITE VIOLATION IN SYSTEM CALL	Operating system error on read-only file.	-	Op System

TABLE B-4. SYSTEM REQUEST LANGUAGE ERROR MESSAGES

Error Code	Message	Significance	Issuing Routine
0001 to 0099	ILLEGAL KEYWORD	An illegal keyword was specified. The error number indicates the ordinal of the parameter in the user's calling sequence. Any routine can issue this error.	ALL
0100	PARAMETER MISSING	A required parameter was not specified. Any routine with required parameters can issue this error.	ALL
0104	ILLEGAL PFI ORDINAL	The 'ENTRY=' parameter is 0 or greater then the number of entries in the SRL defined area.	Q5DCDPFI
0105	ILLEGAL ROUTINE USAGE	A decode routine was called prior to the routine which obtains the desired information.	Q5DCDPFI
0200	SRL BUG-ILLEGAL OPTION	SRL specified an illegal option for a system message. Any routine that issues a system message can receive this error.	ALL
0201	SRL BUG-ILLEGAL BETA	SRL supplied a Beta area for a system message which was rejected by the system.	ALL
0202	SRL BUG-UNRECOGNIZED ERROR CODE	Unrecognized R code in an Alpha or an unrecognized SS code in a Beta.	ALL
0300	MORE FILES TO LIST	More files exist than can fit in the SRL defined buffer area.	Q5LIFPRI Q5LFIPUB Q5LFIPOL
0301	ILLEGAL POOL	The caller is not a member of the specified pool or the pool is not attached.	Q5LFIPOL
0320	ILLEGAL MESSAGE LENGTH	Message buffer length too long or too short.	Q5SNDMCR Q5SNDMCE Q5SNDMJC
0321	ILLEGAL DESTINATION	The controllee or controller specified in the request does not exist.	Q5SNDMCR Q5SNDMCE Q5SNDMJC

TABLE B-4. SYSTEM REQUEST LANGUAGE ERROR MESSAGES (Contd)

Error Code	Message	Significance	Issuing Routine
0322	LOGGED OUT TERMINAL	The controller specified in the request is a logged out terminal.	Q5SNDMCR
0323	DIFFERENT SUFFIX	The controller specified in the request is a terminal which is now logged on under a different suffix.	Q5SNDMCR Q5SNDMJC
0324	SYSTEM BUFFER BUSY	The system buffer is busy. Try again later.	Q5SNDMCR Q5SNDMJC
0325	CONTROLLEE BUSY	The controllee which is the destination of the message already has text from a controller.	Q5SNDMCE
0340	ILLEGAL BUFFER LENGTH	The user's message buffer is too short or too long.	Q5GETMCR Q5GETMOP Q5GETMCE
0341	NO MESSAGE AVAILABLE	No message is available for this request.	Q5GETMCR Q5GETMOP Q5GETMCE
0342	ILLEGAL MESSAGE LENGTH	Count of bytes returned was zero or greater than 4096.	Q5GETMCR Q5GETMOP Q5GETMCE
0343	LEVEL 1 TASK	The caller is a level 1 task and therefore cannot have a controller from which to obtain a message.	Q5GETMCR
0344	CONTROLLER MESSAGE WAITING	A message cannot be transmitted because the task has a message from a controller waiting.	Q5GETMCE
0345	CONTROLLEE WAITING FOR MESSAGE	The controllee from which a message is expected is waiting.	Q5GETMCE
0350	CONTROLLEE ALREADY EXISTS	The controllee cannot be initiated because a controllee already exists.	Q5INIT
0351	CONTROLLEE filename NOT FOUND	The controllee file (filename) does not exist.	Q5INIT
0352	NOT ENOUGH TIME	There is insufficient time to run the controllee.	Q5INIT
0353	ILLEGAL PRIORITY	An illegal priority value was specified.	Q5INIT
0354	DROPFILE CREATE ERROR	An error was encountered when attempting to create the dropfile.	Q5INIT
0355	filename NOT EXECUTABLE	The controllee program file (filename) is not executable.	Q5INIT
0356	filename IO ERROR	A mass storage error was encountered when attemping to read the controllee program file (filename).	Q5INIT
0357	SYSTEM TABLES FULL	Controllee cannot be initiated because the system tables are full. Try again later.	Q5INIT
0358	filename ABNORMALITY	The controllee file (filename) cannot be initiated because of an abnormaility in the file or in the dropfile I/O number.	Q5INIT
0359	TOO MANY LEVELS	Controllee cannot be initiated because five levels of controllee tasks already exist.	Q5INIT
0360	DROPFILE TOO SMALL	Controllee cannot be initiated because the drop file is too small.	Q5INIT

TABLE B-4. SYSTEM REQUEST LANGUAGE ERROR MESSAGES (Contd)

Error Code	Message	Significance	Issuing Routine
0361	PERSISTENT DROPFILE	System unable to destroy existing dropfile.	Q5INIT
0362	INTERRUPT TABLE FULL	Controllee cannot be restarted because interrupt Register Table is full. Try again later.	Q5INIT
0363	DROPFILE VERIFY ERROR	Controllee cannot be initiated because the drop-file cannot be verified.	Q5INIT
0364	DISK READ BUFFER FULL	Insufficient system buffer space to initiate task. Try again later.	Q5INIT
0365	filename BAD MINUS PAGE	The controllee file (filename) contains a bad minus page.	Q5INIT
0366	SYSTEM BUG-DROPFILE VERIFICATION	System detected an undefined error in drop file verification. System error.	Q5INIT
0367	PRIVILEGED OPEN	Controllee program file is currently open (using a privileged OPEN) to a privileged user.	Q5INIT
0370	NO CONTROLLEE TO DISCONNECT	No controllee exists to disconnect.	Q5TERMCE

GLOSSARY C

Absolute Binary Card -
See 80-Column Binary Card.

Access -
A parameter that specifies the read or write access desired. The system grants access only if the lockout field of the file index table allows such access.

Access Station -
A CYBER computer system used to enter jobs into the CYBER 200 system and to control peripheral devices.

Account Identifier -
1 through 8 characters indicating who is to be charged for system resources attributable to a user number.

Batch Deck -
A card deck that begins with a STORE card and that ends with a card having the digits 6, 7, 8, and 9 multipunched in column 1.

Batch Job -
A part of a batch deck that is executed under control of the batch processor as a separate entity. A batch job begins with a job statement.

Batch Pro -
See Batch Processor.

Batch Processor -
A system utility that processes batch jobs. Control statements in the job having file names are executed as controllees of the batch processor.

Byte -
A sequence of eight bits that is a subdivision of a word and is sufficient to represent a single character.

Card Reader ID Card -
Another name for STORE card.

Checkpoint -
A system feature that captures a task and any of its controllees at some point into execution such that the task can be restarted from that point. Checkpoint is called through a FORTRAN program by the name CHKPNT.

Controllee -
A task called into execution by a controller.

Controllee Chain -
A linked series of tasks that results when one task brings another task into execution. That task can, in turn, initiate another task. As many as five levels of tasks can be involved. The first level is level 1; the last is level 5.

Controllee File -
See Virtual Code File.

Controller -
A relative term that indicates a member of a controllee chain has a controllee task attached. A controller might be a controllee of another task.

CPU -
Central processing unit, the computational facility of the CYBER 200 system.

Dayfile -
A file produced by the batch processor for a batch job that gives a history of the job. Information on the file includes the time various control statements began execution and any error or status information produced by system utilities. The dayfile is printed as part of job output.

Directive -
Supplementary control information in a file required in addition to a utility call. Directives are required, for example, with UPDATE and TCOPY.

Drop File -
The file management category of a file created by the system to contain modified pages of an executing task, free space, and write-temporary files.

Drop file names are formed by the system shifting the controllee file name right two characters and prefixing it with digits that identify the suffix (digits 1 through 4 corresponding to suffixes A through D) and the level 1 through 5 in a controllee chain.

Explicit Input/Output -
A means of accessing a mass storage or tape file in which data is buffered under program control.

Family File -
A set of files with names that begin with Pnn which is printed at job termination when a file with a name PXX is added to the family.

File -
A collection of data that can be accessed by file name. In the absence of an adjective such as card or tape, all references to files in this manual imply mass storage files.

File Index Table -
A system table that holds all information relating to file characteristics. Output from the AUDIT or FILES utilities shows much of the table information.

File Type -
A category that defines file structure from a system standpoint. File types are physical data and virtual code.

Implicit Input/Output -
A means of accessing a mass storage file in which the system brings a page of the file into main memory in response to a reference on that page.

Input/Output Connector (IOC) -
An entry in a minus page that links a file with a task.

Invisible Package -
A hardware convention that contains the address and control information for the corresponding job.

Job -
A part of a batch deck that is to be executed under control of the batch processor. A job begins with a job card and ends with a card having the digits 6, 7, and 9 multipunched in column 1.

Labeled Tape -
A magnetic tape with labels conforming to American National Standard X3.27-1969, Magnetic Tape Labels for Information Interchange.

Large Page -
128 small pages; 65 526 contiguous words of 64 bits.

Library -
A file of modules in a format produced by OLE that can be used to satisfy external references during loading.

Link Station -
A CYBER computer system used to enter jobs into the CYBER 200 system and to control peripheral devices.

Local File -
A private file that is destroyed by the system after termination of the batch job or terminal session that created the file.

Main Memory -
Memory associated with the central processing unit from which instructions can be executed. Also called MCS (Magnetic Core Storage).

Management -
A category that defines whether a file is to receive special processing after task execution. Management categories are mass storage, output, scratch, write-temporary, and system-generated or user-generated drop files.

Map -
Part of the minus page of a virtual file that relates virtual addresses with physical mass storage addresses.

Mass Storage -
(1) In a general sense, mass storage indicates disk-resident. (2) A file management category that indicates no special processing after task termination.

Minus Page -
The first page of a virtual file used by the system to hold items such as the invisible package, input/output connector information, and maps of defined virtual space. Drop files contain a second minus page.

Nonprivileged -
A status that allows access to files owned by the same user number under which the task is running, to public files, and to authorized pool files.

Object Code File -
A file generated by compilation or assembly of a source language program that can be used by the loader to produce an executable file.

Ole -
System utility that creates and modifies a file in library format or modmerge file format.

Output File -
A file management category that indicates a file is destined for print or punch equipment.

Also, a generic term for a file being written, as opposed to an input file being read.

Ownership -
A category for each file that determines what nonprivileged tasks can access a mass storage file. Ownership categories are private, pool, and public. Private includes local and permanent files.

Pack File Index (PFI) -
A table of 16-word file index table entries that exists on each pack to control the files located on that pack.

Page -
The unit by which main memory is managed; a block of contiguous 64-bit words. Can be a large page or small page.

Page Fault -
Reference by virtual address to a page not currently in main memory, causing a program interrupt and paging in.

Page Zero -
The second page of a virtual code file into which the CPU stores the task's register file when the task is not in the CPU.

Permanent File -
A private file that remains in the system after termination of the batch job or interactive session that creates it.

Physical Memory Address -
Address of a page in main memory. Also called physical address.

Pool -
A set of files created and maintained by a pool boss. More than one user number can access a pool.

Pool File -
An ownership category that indicates a file can be accessed by any privileged task and by any task running under a user number the pool boss authorizes.

Private File -
An ownership category that indicates a file can be accessed only by a task running under the user number under which the file is stored.

Privileged -
A status that allows access to all files in the system (except local files belonging to other users) and to most operating system functions.

Public File -
An ownership category that indicates a file can be accessed by all users.

Scratch -
A management category that indicates a file is to be destroyed upon termination of the task that created it.

Security Level -
A level 0 through 7 established when a file is created. A user number is also associated with a security level. A user cannot access a file with a higher security level.

Small Page -
512 contiguous 64-bit words.

Source File -
(1) A generic term for a file containing information used by a utility or other task whose specific meaning depends on the context of its use: the controllee file associated with a drop file, for instance, is termed the source file.

(2) In an UPDATE utility context, a file produced by UPDATE that would allow re-creation of a new program library on a subsequent creation run.

Suffix -
A letter A, B, C, or D that is associated with a user number during task execution. All batch jobs execute under suffix D; interactive tasks execute under the suffix specified by LOGON.

System Billing Unit (SBU) -
An installation-defined unit used for charging system resources. The unit might incorporate tape use/access, number of tape functions, number of disk accesses, number of pages transferred to or from disk, and CPU usage in microseconds. An example of SBU is time in microseconds of CPU use.

System Message -
The means by which the operating system and user tasks communicate with each other. System messages, which are formatted in Alpha words and Beta words, are described in volume 2 of the operating system reference manual.

System Time Unit -
An installation-defined unit used for allocating system resources. The unit might incorporate tape use/access, number of tape functions, number of disk accesses, number of pages transferred to or from disk, and CPU usage in microseconds. An example of STU is time in microseconds of CPU use.

Task -
An executable program.

User Number -
Six digits that identify a file owner or user of system resources. Only one task can be in execution for a given user number and suffix at one time.

User1 -
A generic name for system routines that process card decks, print files, and punch files.

Virtual Address -
Address that refers to virtual memory and is translated, through the page table, into a physical address.

Virtual Code File -
A file type that indicates an executable file having a minus page as its first page and a page zero as its second page. The file must be created by the loader. A virtual code file is also called a controllee file. Contrast with Object Code File.

Virtual Memory -
A concept by which physical main memory can be addressed as if it were as large as needed.

Word -
A division of main memory or mass storage corresponding to 64 bits. Bits are numbered 0 through 63 left to right.

80-Column Binary Card -
A punch card that is a representation of fifteen 64-bit words. No conversion occurs during card input or output. Also called an absolute binary card.

CONTROL STATEMENT SUMMARY

ATTACH, { lfn-list / * }.

AUDIT, PN=pkid-list, PF=lfn-list, UN=userno, PL=pl-list, OP=opts, DT=mmddyy, TM=hhmm, LO=x, OU=lfn/len/dc.

COMMENT. message

COMPARE, alfn, blfn, L=len, A=aadr, B=badr, N=lt.

COPY, inlfn, outlfn, L=len, I=inadr, O=outadr.

DEFINE, lfn/len, ACCESS=acs, TYPE=typ, SECURITY=lvl, PACK=packid, NOEXTEND, NOSEGMENT.

DEBUG, fname, I=iname/iioc, O=oname/oioc/olen.

DUMP, dropfile.

DUMPF, DD=device, VSN=id-list, DE=density, RE=days, UN=userno, PL=pl-list, PN=pkid-list, PF=lfn-list, OP=opts, DT=mmddyy, TM=hhmm, LO=x, OU=lfn/len/dc.

EDITPUB, { L / D=lfn-list } , N=lfn-list,P=lfn-list, VRI=vri.

EXIT.

FILES, lfn-list, PUBLIC= { ALL / lfn-list }, PRIVATE={ ALL / lfn-list } , POOL=poolname, lfn-list, L=lfn.

GIVE, { =ALL / lfn-list }, { U=newown / P=poolname, SHARE=perm } .

jobname, Tt, TVvalue.

LOAD, lfn-list, CNTROLEE=lfn/len, CDF=dlen, OUTPUT=lfn/len, LIBRARY=lib-list, EQUATE=sub,name, ENTRY=ept, DEBUG=mod-list, VR=string, ORIGIN=bitadr, GRSP=list,bitadr, GRLP=list,bitadr, GROS=com-list,bitadr, GROL=com-list,bitadr, GRLPALL= , DSA=bitadr.

LOADPF, DD=device, VSN=id-list, DE=density, UN=userno, PL=pl-list, PF=lfn-list, OP=opts, DT=mmddyy, TM=hhmm, PN=pkid-list, LO=x, OU=lfn/len/dc.

LOOK, fname, I=iname, L=oname/olen/disp.

NORERUN.

OLE, INPUT=lfn-list, { NEWLIB=liblfn / MODMERGE=modlfn } , OMIT=sfn, mod-list, LIST=opt, OUTPUT=lfn/len.

PACCESS, poolname, USER=userno.

PATTACH, poolname.

PCREATE, poolname.

PDELETE, poolname, USER=userno.

PDESTROY, poolname.

PDETACH, poolname.

PFILES, { poolname / USER=userno } .

PURGE, lfn-list, CL=pool, ST=xxx.

READCC, lfn.

REQUEST, lfn/len, ACCESS=acs, TYPE=typ, SECURITY=lvl, PACK=packid, NOEXTEND, NOSEGMENT.

RERUN.

RETURN, { lfn-list / * } .

ROUTE, lfn, DC=dc, { DEF / SAVE, } , IC=ic, FID=fffff, EC=ec, CM=cm, ST=st, TID=yyyyyyy, OT=ot, DI=iiiiiii.

SWITCH, oldlfn, newlfn, TYPE=typ, ACCESS=acs, RETENTION=days, DROP=dlen.

TCOPY, lfn.

TV, value+

UPDATE, { C / C=file } , D,F, { I / I=lfn } , L=opt, { N / N=lfn/#nnn } , { O / O=lfn/#nnn } , { P / P=lfn } , { S / S=lfn/#nnn } ,
{ T / T=lfn/#nnn } , 8, *=c, /=c.

INDEX

CONTROL DATA CORPORATION

COMMENT SHEET

MANUAL TITLE: CDC® CYBER 200 Operating System 1.4
Reference Manual Vol. 1 (of 2)
PUBLICATION NO.: 60457000 **REVISION:** A

NAME: _____

COMPANY: _____

STREET ADDRESS: _____

CITY: _____ **STATE:** _____ **ZIP CODE:** _____

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

CUT ALONG LINE

AA3419 REV. 4/79 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

TAPE

FOLD FOLD

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
4201 Lexington Avenue North
Arden Hills, Minnesota 55112

CUT ALONG LINE

FOLD FOLD

TAPE TAPE

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION